TRABALHO DE GRADUAÇÃO

Localização e Mapeamento de VANT quadrirrotor INDOOR

Jessé Barreto de Barros

Brasília, Dezembro de 2016

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Localização e Mapeamento de VANT quadrirrotor INDOOR

Jessé Barreto de Barros

Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Carlos H. Llanos Quintero, ENM/UnB _____
Orientador

Prof. José Maurício S. T. da Motta, ENM/UnB _____
Examinador Interno

Prof. Renato Coral Sampaio, FGA/UnB _____
Examinador Interno

Brasília, Dezembro de 2016

FICHA CATALOGRÁFICA

JESSÉ, BARRETO DE BARROS
Localização e Mapeamento de VANT quadrirrotor INDOOR

[Distrito Federal] 2016.

x, 70p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. QUADRIRROTOR 2.SLAM
3. VANT 4.ROBÓTICA

I. Mecatrônica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

BARROS, J. B., (2016). Localização e Mapeamento de VANT quadrirrotor INDOOR, (2016). Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$038/2016, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 70p.

CESSÃO DE DIREITOS

AUTOR: Jessé Barreto de Barros

TÍTULO: Localização e Mapeamento de VANT quadrirrotor INDOOR.

GRAU: Engenheiro ANO: 2016

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Jessé Barreto de Barros

SQN 408 Bloco G, nº 104, Asa Norte.

70856-070 Brasília – DF – Brasil.

Dedicatória

Dedico esse trabalho à minha família e amigos.

Jessé Barreto de Barros

Agradecimentos

Agradeço à minha avó, Maria Adail Nunes Barreto, que sempre me auxiliou e me instruiu a seguir os meus sonhos e enfatizou a importância de estudar e que apesar de todos os seus problemas de saúde ainda é uma pessoa forte e que por toda a minha vida será considerada como um exemplo de caráter, perseverança e idoneidade.

Agradeço à minha mãe, pai e irmã que me apoiaram para vencer mais um desafio e que me dão a confiança por saber que, apesar dos revezes da vida, eu sempre terei o apoio deles.

Agradeço aos grandes amigos que adquiri durante os anos na UnB e que tive a felicidade de compartilhar muitas alegrias. Dentre os amigos, quero agradecer especialmente ao Marco Emílio (Marquemilho), Rodrigo Teixeira, Daniel Vincentin, Victor Hugo Costa, André Almeida (Andrezinho), Filipe Caixeta e Geordana Maeda.

Agradeço à minha querida namorada, Chanhui Li, que é a minha melhor amiga, companheira e conselheira. Sendo a pessoa responsável por me fazer sorrir mesmo quando tudo parece dar errado e que compartilha comigo planos e sonhos.

Por último, mas não menos importante, agradeço ao meu professor Carlos Humberto Llanos pela oportunidade e orientação e também ao professor Renato Coral que é um professional a ser seguido de exemplo. A oportunidade de trabalhar no LEIA com esses professores para mim foi excepcional e que ao meu ver faz da frase dita por Newton uma realidade, "Se vi mais longe foi por estar de pé sobre ombros de gigantes".

Jessé Barreto de Barros

RESUMO

Este trabalho possui como objetivo a implementação de uma técnica para a localização e o mapeamento simultâneos de um robô aéreo (VANT) do tipo quadrirrotor dentro de um local de ambiente fechado (indoor), através da utilização dos dados provenientes dos seus sensores inerciais e das imagens obtidas pelas suas câmeras embutidas. O projeto foi baseado na utilização de uma técnica para a localização e o mapeamento visual monocular (PTAM) aliado à um filtro de kalman estendido para efetuar a fusão de dados e predizer a posição do VANT em tempo real. O desenvolvimento foi feito utilizando recursos da plataforma ROS. Foram efetuados experimentos com um AR.Drone como VANT e com o ambiente de simulação Gazebo que é compatível para utilizar recursos do ROS. Adicionalmente, buscou-se a utilização de uma placa de desenvolvimento do tipo SoC para atuar como computador remoto para processamento dos algoritmos do sistema. Espera-se que esse trabalho venha contribuir com a adição de conhecimentos sobre plataforma ROS para o laboratório LEIA e também a utilização de técnicas de localização e mapeamento para robôs móveis.

Palavras Chave: QUADRIRROTOR, MAPEAMENTO, LOCALIZAÇÃO, VANT, ROBÓTICA

ABSTRACT

This work has as goal the development of a method for the simultaneous localization and mapping of an aerial robot (UAV) quadrotor for an indoor application, using data obtained from its inertial sensors and the images originated from its cameras. The project was based on a monocular SLAM technic, called PTAM, together with an extended kalman filter (EKF) for sensor fusion and prediction of the UAV position on real time. The development was made using resources from ROS. Tests were made using both an AR.Drone UAV and with simulations using Gazebo, a simulation framework compatible with ROS resources. Additionally, it was attempted to use a development board SoCKit to be used as a remote computer to process the system's algorithms. This work looks forward to add to the laboratory LEIA's knowledge pool familiarization with the ROS framework and, also, the use of new techniques for mobile robots simultaneous localization and mapping (SLAM).

Keywords: QUADROTOR, SLAM, UAV, ROBOTICS

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

O	Sistema de Coordenadas	
R	Matriz de Rotação	
f	Força	[N]
m	Massa	[kg]
g	Módulo da Aceleração Gravitacional	$[m/s^2]$
A	Componente da Força de Arrasto	[N]
C	Força de Coriolis	[N]
t	Instante de Tempo	[s]
u	Vetor de Variavéis de Controle	
$X(t)$	Vetor de Espaço de Estados	
p	Vetor de Posição 3D	
v	Velocidade linear	[m/s]

Símbolos Gregos

ϕ	Ângulo de Rotação no eixo x - Rolagem	[°]
θ	Ângulo de Rotação no eixo y - Arfagem	[°]
ψ	Ângulo de Rotação no eixo z - Guinada	[°]
Ω	Velocidade Angular	[rad/s]
η	Vetor de Orientação	[rad]
τ	Torque	$[N \cdot m]$
ω	Distorção Radial da Câmera	[rad]

Grupos Adimensionais

i, j	Contador

Subscritos

ref	referência

Sobrescritos

$\dot{}$	Variação temporal
$\bar{}$	Valor medio
$\hat{}$	Valor Previsto

Siglas

VANT	Veículo Aéreo Não-Tripulado
UAV	*Unmanned Aerial Vehicle*
VTOL	*Vertical Take-Off and Landing*
SLAM	*Simultaneous Localization and Mapping*
PC	*Personal Computer*
AR	*Aumengted Reality*
EPP	*Expanded Polypropylene*
DC	*Direct Current*
LED	*Light Emmiter Diode*
PWM	*Pulse Width Modulation*
DDR	*Double Data Rate*
RAM	*Synchronous Random Access Memory*
DSP	*Digital Signal Processor*
DHCP	*Dynamic Host Configuration Protocol*
UDP	*User Datagram Protocol*
IMU	*Inertial Measurement Unit*
MEMS	*Micro Electro-Mechanical System*
DOF	*Degrees of Freedom*
CMOS	*Complementary Metal-Oxide Semiconductor*
VGA	*Video Graphics Array*
QCIF	*Quarter Common Intermediate Format ou Quarter Common Interchange Format*
ROS	*Robotic Operating System*
OSRF	*Open Source Robotics Foundation*
SoC	*System on Chip*
FPGA	*Field Programmable Gate Array*
HPS	*Hard Processor System*
LE	*Logical Elements*
PLL	*Phase-Locked Loop Basic*
SDRAM	*Synchronous Dynamic Random Access Memory*
SD	*Secure Digital*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
SSH	*Secure Shell*
CPU	*Central Processing Unit*
LEIA	*Laboratory of Embedded Systems and Integrated Circuits Applications*
GRACO	Grupo de Automação e Controle da Universidade de Brasília
LARA	Laboratório de Automação e Robótica

Capítulo 1

Introdução

1.1 Aspectos Gerais

Esse capítulo está organizado pela contextualização do tema e oque já foi desenvolvido na área de quadrirrotores como veículos aéreos não tripulados e introduz os objetivos e alguns conceitos utilizados durante esse trabalho.

As áreas de pesquisa apresentadas durante o trabalho serão expostas com maior riqueza de detalhes conforme a necessidade de uma maior compreensão dos assuntos para o entendimento e o desenvolvimento do trabalho.

1.2 Contextualização

O avanço constante e crescente na computação, aliada ao aumento na tecnologia na, produção e a miniaturização de componentes eletrônicos possibilitou a implementação e utilização de técnicas de controle em dispositivos automatizados principalmente com grandes reflexos na área da robótica. Nesse contexto, a maturação da robótica pode ser claramente visualizada pelo grande número de suas aplicações, que são, cada vez mais, presentes e necessárias na nossa sociedade.

Os robôs não estão mais apenas no nosso imaginário e na ficção científica, sendo que esta última também contribui para o desenvolvimento de conceitos no contexto de ética na citada área, com obras literárias como, por exemplo, Eu, Robô de Isaac Asimov [11]. Podemos visualizar o crescente papel da robótica e da automação, sendo inserida em nosso cotidiano com o desenvolvimento de robôs que nos auxiliam na limpeza de nossos lares [12], e em um futuro próximo na área automotiva, possibilitando dirigir veículos em rodovias [13] [14].

Nos últimos anos tem crescido o interesse e o desenvolvimento de veículos aéreos não-tripulados (VANTs), também conhecidos da sigla em inglês UAV, na área da robótica aérea com aplicações nas áreas de segurança e vigilância [15], de inspeção de instalações industriais [16] e de transporte de cargas [17]. Neste caso, essas aeronaves podem ser autônomas ou remotamente controladas.

Atualmente há diversas classes de VANTs e, os mesmos, podem ser divididos por possuírem

estruturas de chassi com asas fixas, como as de aviões comerciais para transporte de passageiros, ou com asas giratórias, como as de helicópteros. A classe de asas fixas é bem conhecida por serem utilizadas em aplicações militares como as dos primeiros Drones da força aérea dos Estados Unidos, vide o presente na Figura 1.1). Para fins civis e comerciais, a grande maioria dos VANTs possuem asas giratórias, tais como as de helicópteros para pouso e decolagens verticais (VTOL), bem como os VANTs do tipo quadrirrotor, sendo este último caso o predominante em aeronaves comerciais de baixo custo.

Figura 1.1: VANT militar utilizado pela força aérea dos Estados Unidos do tipo de asas rígidas. [1].

1.2.1 Quadrirrotores

Os quadrirrotores são, geralmente, formados por um chassi em formato de cruz que suporta os seus quatro motores. Apesar da semelhança com os helicópteros, os quadrirrotores são veículos completamente diferentes. Essa diferença é ainda mais evidenciada quanto à forma em que ambos os veículos são controlados. Um helicóptero se movimenta através da modificação do ângulo de ataque das pás de suas hélices. Neste caso, o ângulo de ataque é o ângulo formado entre o fluído em movimento (através das pás das hélices) e uma linha imaginária que secciona a hélice ao meio, conforme a Figura 1.2. Já o quadrirrotores não podem fazer o mesmo, porque possuem os ângulos de ataque de suas hélices fixos, e o seu movimento é obtido atráves da diferença na velocidade de giro dos seus propulsores.

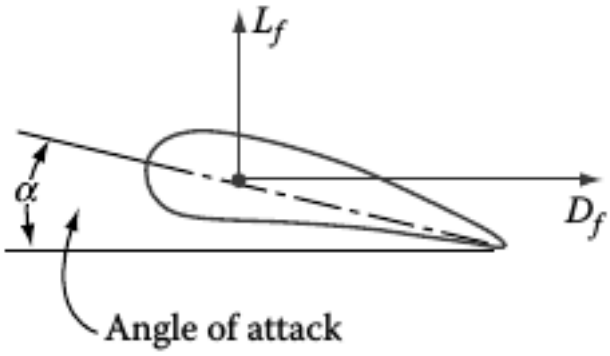

Figura 1.2: Representação de uma pá de hélice com destaque ao ângulo α que o ángulo entre o movimento do fluido e a linha que secciona a pá da hélice. L_f é a força de sustentação e D_f é a força de arrasto que são aplicadas sobre a hélice. [2].

Os VANTs do tipo quadrirrotores possuem diversas vantagens em relação aos de asas fixas. Eles podem se movimentar em qualquer direção e também conseguem pairar e voar em baixas velocidades. Em adição às suas características VTOL que permitem o pouso e decolagem em diferentes tipos de terrenos, enquanto as aeronaves de asas fixas necessitam de uma pista de pouso própria. Um exemplo de quadrirrotor UAV está presente na Figura 1.3.

Figura 1.3: VANT quadrirrotor comercial MD4-1000 da microdrones [3].

1.3 Definição do problema

O uso de quadrirrotores na robótica móvel é um assunto bem estabelecido e crescente no meio acadêmico, graças às suas vantagens em termos de miniaturização. Atualmente, os quadrirrotores comerciais de pequeno porte possuem tamanhos reduzidos Figura 1.4, se comparados com os VANTs militares, possuindo (adicionalmente) a capacidade de vôo em locais fechados e com pouco espaço por serem veículos aéreos VTOL.

Porém, os desafios dessa área são grandes devido a baixa capacidade das baterias, a reduzida quantidade de sensores que podem ser utilizados, assim como o limitado poder computacional embarcado, junto com a necessidade de estabilização devido à natureza complexa da dinâmica desses veículos.

Esses problemas são agravados ainda mais devido a pequena quantidade de carga que os drones de uso comercial de baixo custo suportam. Qualquer adição à carga carregada pelo robô implica em um tempo de voo ainda mais reduzido.

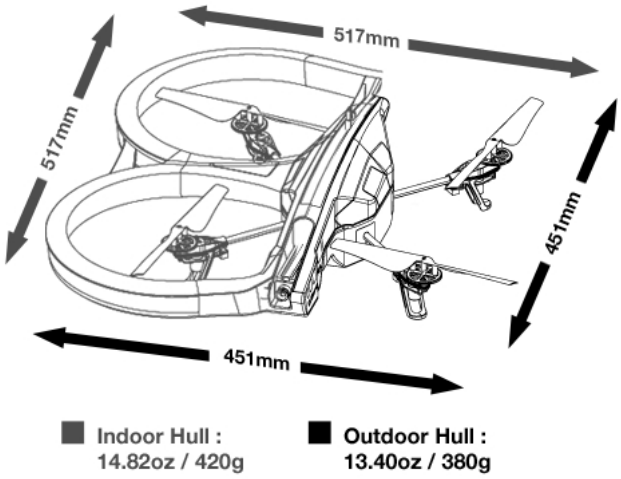

Figura 1.4: Dimensão física do VANT comercial de baixo custo que foi utilizado nesse trabalho. Com a comparação de seus cascos de proteção disponíveis pelo fabricante [4].

No contexto da robótica móvel é essencial a obtenção de um conhecimento mínimo do ambiente em que o robô está se locomovendo, a fim de poder planejar uma rota mais eficiente para diversas aplicações. Por exemplo, objetivando aumentar o tempo de uso das suas baterias e evitar rotas que comprometam a estrutura do robô, como escadas para robôs humanoides ou áreas onde o robô possa ficar preso Figura 1.5.

Porém, são diversas as situações em que não é possível obter um mapa prévio, sendo que o robô móvel deve ser capaz de se adaptar a essas situações e, automaticamente, mapear o ambiente a sua volta. Mas, o mapeamento sozinho não adianta muito se o robô não conhece a sua posição atual (no mapa que está sendo desenvolvido), sendo por isso necessário se dispor de um processo de localização do robô no mapa. Mas para poder saber a sua posição no mapa, é necessário um conhecimento prévio do mapa.

Baseado nesse contexto (de um problema que parece ser quase que paradoxal), deu-se a origem de técnicas e algoritmos de SLAM (*Simultâneous Localization and Mapping*).

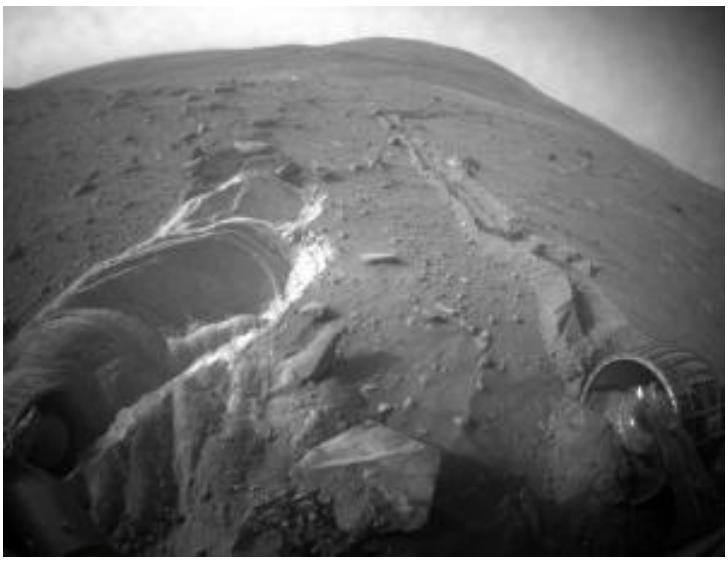

Figura 1.5: Robô *Spirit* lançado pela NASA para explorar a superfície de Marte ficou com as suas rodas presas na areia fina [5].

Com o intuito de navegar o VANT de forma autônoma no interior de um laboratório (onde o ambiente é desconhecido e para ser explorado pelo robô), o uso de técnicas de localização e mapeamento de um quadrirrotor em um ambiente interno é essencial. Porém devido ao alto custo computacional desses tipos de algoritmos, é necessário o uso de um computador processamento remoto para execução dos algoritmos de visão computacional, voltados para o SLAM (Simultaneous Localization and Mapping), e também para o controle do posicionamento do quadrirrotor, adicionando uma malha a mais de controle realimentado, com o desenvolvimento de um controlador, para aumentar a confiabilidade e robustez do sistema como um todo.

Realizar simultaneamente a tarefa de localização e mapeamento é um processo que possui um nível de complexidade alto e demanda o uso de diversos algoritmos intermediários de visão computacional, visando a estimativa confiável da posição do veículo em relação a um mapa, este último obtido pelo uso de algum algoritmo de extração de características da imagem, a fim de serem tratados como pontos de referências (*landmarks*).

Além disso, para aumentar a confiabilidade dos dados obtidos pelas câmeras, aliadas com processamento de imagem, é também necessário unir os dados provenientes dos sensores.

No entanto, devido às incertezas envolvidas nas medições, é necessária a utilização de técnicas de filtragem estocástica, por exemplo, utilizando um filtro de Kalman, para unir os dados dos diferentes sensores presentes no VANT. Esta área é denominada habitualmente como *fusão sensorial*.

1.4 Justificativa do Trabalho

A crescente popularidade do uso de quadrirrotores e a facilidade de obtenção de veículos VANT fazem com que trabalhos nessa área possam servir de base para o desenvolvimento de melhorias em

inúmeros contextos e aplicações que possuem urgência em nossa sociedade, por exemplo, sistemas móveis de vigilância de tráfego, busca por focos de mosquitos da dengue em áreas de díficil acesso [18] e visualização aérea de áreas de desastres naturais.

A utilização de plataformas *open source* como o ROS são essenciais na área da robótica por que possibilitam o uso de um sistema padrão para o fácil compartilhamento de conhecimento entre diferentes instituições de pesquisa e também serve como plataforma para o desenvolvimento de aplicações otimizadas para a robótica por possuir bibliotecas voltadas para a comunicação entre robô, máquinas remotas, sensores e plataformas de simulação.

O ROS também possui a possibilidade de utilizar diversas plataformas de simulação integradas às aplicações que o utilizam como *framework*. Simuladores são aplicações muito utilizados na robótica porque possibilitam a obtenção de dados sobre o robô e a aplicação desenvolvida sem o risco de danificar a plataforma robótica ou, caso a mesma não se encontra disponível ou com problemas, validar conceitos e aplicações.

Portanto, trazer o conhecimento desse tipo de plataforma de desenvolvimento em robótica para laboratórios de pesquisa como o LEIA no GRACO significam aumentar a possibilidade de desenvolver aplicações que possam ser facilmente compartilhadas e citadas, ou desenvolvidas em conjunto, com outros laboratórios tanto dentro da UnB, por exemplo, o LARA, quanto institutos de pesquisa em robótica internacionais.

1.5 Objetivos do Trabalho

1.5.1 Objetivos Gerais

Utilizar a plataforma ROS para desenvolver uma aplicação de localização e mapeamento de um VANT em um local fechado utilizando um VANT comercial de fácil obtenção, o Parrot AR.Drone.

Utilizar como computador embarcado uma placa de desenvolvimento do tipo SoCKit (da Altera) para processar remotamente em *hardware* algoritmos de localização e mapeamento que por sua vez possuem diversos gargalos no processamento utilizando arquiteturas convencionais.

1.5.2 Objetivos Específicos

Os objetivos específicos do projeto foram:

- Aprendizado e aperfeiçoamento no uso da plataforma de robótica ROS.
- Instalação da plataforma ROS em um sistema embarcado do tipo SoC.
- Desenvolvimento de uma aplicação para comunicação com o VANT.
- Calibração da câmera frontal do VANT para utilizar técnicas de odômetria visual para a localização do quadrirrotor.

- Desenvolvimento de uma aplicação para a localização do VANT utilizando os seus sensores embarcados.
- Desenvolvimento de uma aplicação para o mapeamento do VANT utilizando os seus sensores embarcados.
- Teste e simulação das aplicações de localização e mapeamento em um computador remoto em tempo real.
- Teste das aplicações desenvolvidas no computador embarcado do tipo SoC para identificação dos gargalos do sistema.
- Programação do *hardware* do computador embarcado do tipo SoC para reduzir os gargalos do sistema.

1.6 Apresentação do Manuscrito

Este capítulo teve o objetivo de contextualizar esse trabalho de graduação com as suas possíveis aplicações e também explicar os seus objetivos.

O capítulo 2 possui o objetivo de explicar sobre as plataformas de hardware e também apresentar algumas das bases teóricas utilizadas nesse trabalho.

No capítulo 3 é explicado o desenvolvimento do trabalho utilizando os fundamentos apresentados e também a integração das diversas ferramentas utilizadas, além de descrever a arquitetura da implementação do SLAM monocular no AR.Drone.

O capítulo 4 possui um apanhado dos principais resultados obtidos por esse trabalho e algumas análises sobre esses resultados.

Por último, o capítulo 5 apresenta as conclusões deste trabalho além de propostas para trabalhos futuros desenvolvidos a partir deste.

Capítulo 2

Fundamentação Teórica e Plataformas Utilizadas

2.1 Plataforma AR.Drone Parrot 1.0

O AR.Drone (vide Figuras 2.1 e 2.2) está descrito pela sigla AR (em inglês *Aumengted Reality*), é um dos diversos modelos de quadrirrotor presentes no mercado, sendo um dos VANTs desenvolvidos pela empresa parisiense Parrot SA [4].

Este tipo de equipamento é bastante popular, devido ao longo tempo que o produto está no mercado (o AR.Drone 1.0 foi lançado em 2010 [19], ao seu preço reduzido (se comparado com outros VANTs comerciais), e a sua fácil acessibilidade; o qual o torna um dos VANTs comerciais mais conhecidos e propício para o uso acadêmico [7] [20].

Figura 2.1: O quadrirrotor utilizado com o seu chassi externo de proteção para voos em locais fechados (*in-door*) e ao lado o seu chassi para voos externos.

Figura 2.2: O AR.Drone drone desmontado com todos os seus compontentes internos.

A ideia inicial (com a que foi concebido o Parrot AR.Drone) é a de um brinquedo caro, frágil e com uma tecnologia embarcada *Hi-Tech*, com a possibilidade de ser controlado remotamente atráves de um aplicativo em aparelhos celulares *smartphone*.

No entanto, para os entusiastas em robótica e acadêmicos, o AR.Drone é um computador embarcado, executando um sistema operacional Linux dentro de um robô aéreo com a capacidade de comunicação WiFi auxiliado por diversos sensores, incluindo também duas câmeras.

2.1.1 Estrutura

O chassi do quadrirrotor é composto por uma estrutura tubular feita de fibra de carbono e plástico, a fim de torná-la o mais leve possível (vide Figura 2.3).

Figura 2.3: Estrutura tubular de fibra de carbono utilizada como chassi do AR.Drone.

O chassi externo de proteção é feito de um isopor especial EPP (da sigla em inglês, *Expanded Polypropylene* [21]), que é um material resistente, leve e também reciclável, caso descartado. Esse

chassi possui a função de proteger a estrutura do quadrirrotor, assim como suas hélices e motores para voos em ambientes internos, e/ou quedas e colisões acidentais.

A Parrot também disponibiliza (junto do seu quadrirrotor) um chassi mais leve feito do mesmo material, mas que não protege os rotores no caso de eventuais quedas ou colisões. Porém, o mesmo aumenta a vida útil da bateria durante o voo, por ser mais leve, sendo (neste caso) projetado para voos externos onde o risco de colisão com outros objetos é reduzido.

2.1.2 Propulsores

A movimentação do AR.Drone é obtida através de quatro motores propulsores e as suas hélices. Cada propulsor possui um motor DC sem escovas (*brushless*) que pode atingir velocidades de giro de até 28.500 RPM (quando está pairando), o que correspode em 3.300 RPM de rotação nas hélices com o uso de engrenagens redutoras.

Este cenário corresponde um consumo de 14,5 W por propulsor. Cada motor possui um módulo de controle fabricado especialmente para o AR.Drone, composto por um microcontrolador de 8-bits, um conversor analógico digital de 10-bits de resolução, um LED verde e um LED vermelha, para aviso.

A Figura 2.4 possui a imagem do propulsor sem a hélice e engrenagem de redução.

Figura 2.4: Conjunto motor brushless DC e placa controladora do motor. [6].

Adicionalmente, cada motor do AR.Drone possui a sua própria malha de controle de velocidade e recebe comandos de refêrencia de velocidade da placa principal atráves de sinais em PWM.

2.1.3 Bateria

Para a alimentação (tanto da placa com o computador embarcado e dos seus atuadores), o AR.Drone possui uma bateria de polímero de Lítio (LiPo) (vide Figura 2.5), de 3-células com uma capacidade, quando completamente carregada, de 1000 mAh e uma capacidade de descarga de até 10c com uma tensão de saída de 11.1 V. Isto permite ao quadrirrotor voar aproximadamente 12 minutos, sendo necessário cerca de 90 minutos para recarregá-la completamente.

O que torna essa tecnologia de baterias muito utilizada em dispositivos do tipo portáteis e brinquedos rádio controlados (tal como o AR.Parrot) é a sua baixa massa comparada com a quantidade de carga que pode ser suportada, assim como a sua flexibilidade de formatos que

podem ser adotados e por poderem ter altas taxas de descarga.

O desenvolvimento da tecnológia desse tipo de baterias é um dos fatores que tornaram possível o uso acessível de VANTs pequenos e de baixo custo. Porém, também existem desvantagens que esse tipo de bateria possuem. Por exemplo, baterias LiPo são mais caras que outros tipos de baterias, sendo sua vida útil bastante reduzida (cerca de 400 ciclos de carga e descarga) e, quando danificadas, podem entrar em combustão, se forem carregas ou descarregadas excessivamente ou armazenadas de forma imprópria.

Devido a esses problemas de segurança, a bateria LiPo presente no AR.Drone possui um circuito de segurança para carga e descarga, sendo altamente recomendado carregá-las utilizando o carregador original, devido a que o mesmo carrega as 3-celulas internas das batéria de forma balanceada.

Figura 2.5: Bateria do AR.Drone 1.0 com os seus cabos de alimentação do quadrirrotor e dos cabos para a recarga individual das suas células.

2.1.4 Hardware e Software Embarcado

O Parrot AR.Drone 1.0 possui um computador embarcado com arquitetura baseada no ARM9 32-bits (com um clock de 468 MHz), possuindo uma memória de 128 MB (do tipo DDR RAM de 200 MHz). Adicionalmente, para o processamento visual de suas duas câmeras o computador embarcado também possui um DSP de vídeo (TMS320DMC64x de 800 MHz).

O AR.Drone possui um sistema operacional BusyBox baseado no GNU/Linux (kernel 2.5.27). O *firmware* é responsável pela comunicação, pela estabilização e pelo controle do quadrirrotor.

A placa principal do quadrirrotor possui um conector mini-USB para eventuais reparos e/ou modificações do *firmware* de fábrica, possibilitando adicionar novas funcionalidades ao VANT, como, por exemplo, a instalação de um sensor GPS.

O computador no interior do AR.Drone também possui uma placa WiFi 802.11g, a qual permite a comunicação sem fio para o envio e recebimento de dados, sendo também utilizada para o controle do quadrirrotor remotamente, através de dispositivos móveis como smartphones.

Quando ligado, o VANT aparece como um ponto de acesso para uma rede ad-hoc WiFi. Redes ad-hoc são redes descentralizadas que não precisam de infraestrutura prévia, ou seja, cada nó

da rede atua como um roteador [22]. Nessa rede o AR.Drone pode ser acessado via protocolo de comando *telnet* ou através de um servidor DHCP, que se comunica com outros computadores atráves de uma interface de 3 canais. Neste caso, cada canal possui uma porta UDP com uma funcionalidade distinta.

Pelo canal *command* o VANT recebe os comandos de (a) decolar, (b) pousar, (c) mudar as configurações do controlador interno (controle normal ou agressivo), (d) calibrar os sensores, (e) definir os valores de PWM de cada motor individualmente, (f) mudança dos valores de rolagem, arfagem, guinada e comandos de velocidades através do controlador interno. Neste caso, esse canal possui uma taxa de comunicação de 30 Hz.

Pelo canal *navdata* o VANT emite os dados provenientes de seus sensores e também estados em que o seu sistema está; como, por exemplo, voando, pairando, pousado, iniciado, decolando, pousando ou em modo de emergência. Outros dados emitidos pelo canal são as medidas dos sensores, conjunto de IMUs e sensor ultrasom, quantidade de carga da bateria e velocidade estimada de acordo com o solo. A taxa de comunicação desse canal também é de 30 Hz.

Por último, pelo canal *stream* o AR.Drone envia as imagens da camera frontal e vertical. A imagem da câmera frontal antes de ser enviada possui as suas dimensões reduzidas e é comprimida para aumentar a velocidade de transmissão do vídeo. Uma das características negativas do sistema de vídeo do AR.Drone é que o usuário não possui a opção de receber as imagens das duas câmeras como *stream* de vídeo (simultaneamente), devido a que o sistema só possui um encoder de vídeo para transmissão ao vivo de vídeo.

O problema na utilização do protocolo UDP (para os canais de comunicação) é que o UDP é um protocolo que não possui nenhum *handshake* e por isso expõe a conexão à erros como, por exemplo, a perda de pacotes de dados ou a entrega dos pacotes em ordem diferente àquela enviada. No entanto, esse protocolo possui as vantagens de ser rápido e a implementação utiliza menos recursos computacionais do computador embarcado.

2.1.4.1 Sensores de Medição Inercial (IMU) e Sensor Ultrasom

O sensor IMU do AR.Drone é de 6 graus de liberdade (6DOF), o que disponibiliza ao *software* implementado no veículo medidas dos ângulos de rolagem, arfagem e guinada, assim como as diferentes acelerações nos três eixos espaciais. Essas medidas são essenciais para providenciar a estabilização (enquanto o VANT paira) e também auxiliar no controle da movimentação do VANT.

O sensor IMU é um circuito integrado que possuem partes micro eletromecânicas (MEMS). No contexto de MEMS, o AR.Drone possui um acelerômetro de 3 eixos, um giroscópio de 2 eixos para a rolagem e a arfagem e um giroscópio de 1 eixo para a guinada. O acelorômetro de 3 eixos (Bosch BMA150 [23]) possui como saída de dados acelerações relativas a uma queda livre como unidade de aceleração, ou *g-forces*.

O acelerômetro possui em seu interior pequenas massas de prova com placas acopladas, através de pequenas molas que servem de suspensão e que possuem uma posição de referência. Essas placas móveis (combinadas com as placas fixas no interior do dispositivo) formam pequenos capacitores.

Dessa forma, o movimento dessas massas é medido através da variação da capacitância entre essas placas.

Um acelerômetro em repouso relativo à superfície da terra indicará aproxidamente 1g positivo, devido ao sistema de coordenadas do acelerômetro. Em consequência à esse comportamento, o acelerômetro pode também ser utilizado como um inclinômetro.

Os giroscópios do AR.Drone mensuram as velocidades angulares em graus por segundo, baseando-se no pricípio de conservação do momento angular. Para obtenção do ângulo absoluto θ, em qualquer eixo, é necessário que o sinal de velocidade $\dot{\theta}$ seja integrado no tempo.

Esse processo abre espaço para o *offset error* (ajuste de zero) que deriva através do tempo, porque os giroscópios possuem pequenos erros em suas taxas de leitura de velocidades angulares. Isto é devido, principalmente, ao processo de integração que deve ser utilizado para obter os ângulos a cada instante.

No contexto do AR.Drone, os giroscópios são utilizados para medir os ângulos de arfagem e rolagem (um giroscópio do tipo *IDG-500 Dual Axis*). As características técnicas desse giroscópio é que o mesmo possui duas saídas independentes para os seus dois eixos, para movimentos de alta velocidade, com uma faixa de $500°/s$, com sensibilidade de $2.0mV/\circ/s$. Já o giroscópio para a guinada é mais preciso, Epson Toyocom XV-3500CB. Esse giroscópio possui uma faixa menor de medida, de $100°/s$, e uma sensibilidade de $0.67mV/°/s$.

O AR.Drone também possui um sensor ultrasom em sua parte inferior, o qual consegue medir a altura do VANT com o solo, sendo utilizado no controle de estabilização de altura e também mensura a velocidade relativa vertical do VANT em relação ao solo.

O sensor ultrassônico apesar de essencial para o VANT possui algumas desvantagens em seu uso, que são inerentes ao próprio princípio físico que o sensor utiliza. Em [24] é demonstrado as dificuldades do uso da técnica de ultrasom para o desvio de obstáculos, assim como a degradação da onda de ultrasom, dependendo da superfície que a mesma está sendo refletida.

O sensor ultrassônico instalado no AR.Drone possui uma faixa de operação efetiva de 40 cm até 6 m, porém o mesmo possui muitas dificuldades para obter informações precisas da direção em que o AR.Drone está se movendo em relação ao solo. Em consequência das propriedades físicas da propagação sonora, o ultrasom emitido pelo sensor se propaga em um cone com ângulos de abertura entre 20 à 40 °. Dessa forma o sensor adquire informação de distância de uma superfície inteira, em vez de um conjunto de pontos discretos de informação.

Na Figura 2.6, pode ser visualizada a placa de navegação do VANT. Essa placa possui separadamente a placa principal, com os processadores embarcados e todos os sensores.

Figura 2.6: Vista superior da placa principal do AR.Drone.

2.1.5 Câmeras

O Parrot AR.Drone é equipado com duas câmeras CMOS. Uma instalada horizontalmente na parte frontal do chassi do VANT e a outra instalada na parte inferior do mesmo.

A câmera frontal possui uma resolução VGA 640x480 pixels e uma abertura de 93° de sua lente e com frequências de amostragem de 15 *frames* por segundo.

De acordo com o fabricante, o objetivo inicial dessa câmera seria a observação de outros AR.Drones em voo pelos pilotos, utilizando o aplicativo no celular e, através desse *feedback* visual, participar de um jogo de perseguição entre os múltiplos AR.Drone.

A câmera inferior (que sempre está apontada para o solo quando o AR.DRone está pairando) possui uma resolução de apenas 176x144 pixels QCIF e uma abertura de 64°. Porém a frequência de amostragem dessa câmera é de 60 frames por segundo, sendo que essa frequência de amostragem é maior para reduzir o efeito de borramento *motion blur*, quando as imagens são obtidas com o VANT em movimento.

Além disso, o AR.Drone utiliza as imagens dessa câmera para obter uma estimativa das velocidades horizontais do VANT no *plano-xy*, para fins de auxiliar na estabilização horizontal do VANT. Essas estimações das velocidades horizontais são obtidas através da fusão de dados entre as estimativas das velocidades horizontais obtidas com os IMU e também pela estimativa de velocidade obtida através do cálculo do fluxo ótico das imagens. A Figura 2.7 mostra os resultados obtidos com essa estimativa.

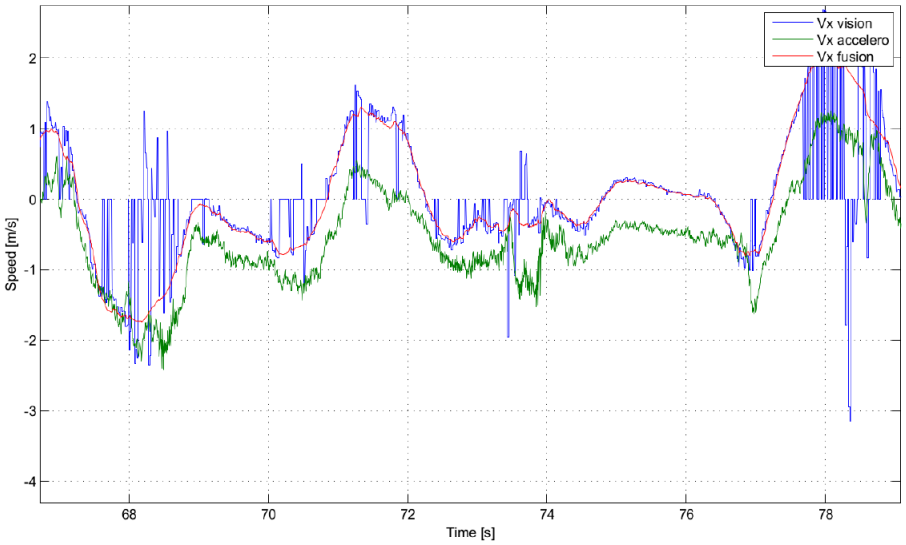

Figura 2.7: Estimativas de velocidade obtidas pelo processamento embarcado no AR.Drone, fonte [7].

A metodologia implementada no VANT para o cálculo do fluxo ótico utiliza o algoritmo proposto por Lucas-Kanade [25], que obtem os deslocamentos das imagens entre diferentes cenas. Em uma explicação rápida, o algoritmo funciona utilizando diferentes subimagens da imagem inicial, e depois calcula o deslocamento dessas subimagens entre as diferentes cenas, através do cálculo da soma dos quadrados das diferenças de intensidade entre os seus pixels, utilizando (posteriormente) o algoritmo de Horn-Schunck [26] para determinar o fluxo de movimento dos pixels na imagem.

Por causa do uso do algoritmo de Lucas-Kanade, variações nos ângulos de rolagem e arfagem não afetam drasticamente a estimação de velocidade do VANT. Porém, a implementação utilizada é menos robusta em cenas com menor contraste. Por isso, a estimação das velocidades no *plano-xy* do AR.Drone dependem fortemente dos contrastes do solo.

Apesar da alta taxa de atualização da câmera inferior, as imagens são fornecidas via WiFi pelo AR.Drone na mesma frequência da câmera frontal (ou seja, 15 Hz).

2.2 A Plataforma ROS

A ROS (*Robotic Operating System*) [27] é uma plataforma de *software* desenvolvido pela OSRF (Open Source Robotics Foundation). A ROS funciona como uma plataforma (*framework*) para o desenvolvimento e uso de *software* voltados para a robótica. O maior pressuposto para a criação da ROS é o compartilhamento, o reuso e a manutenção de códigos, ferramentas, *drivers*, assim como bibliotecas que possam ser integradas para a simplificação no desenvolvimento de tarefas complexas e robustas na área da robótica, e que possam funcionar em uma grande variedade de ambientes e sistemas robóticos.

A ROS foi desenvolvida como uma camada de comunicação em rede *peer-to-peer* (ponto a ponto) entre diferentes processos. Isso facilita o desenvolvimento e a integração de diferentes

bibliotecas e aplicações. Também, devido a esse fato, várias aplicações voltadas para a área da robótica estão utilizando recursos oferecidos pela ROS, ou sendo portadas para funcionar como pacotes suportados pelo sistema da ROS.

Na ROS o modelo de comunicação entre os processos do robô são denominados *nós* (*nodes*) e o canal de comunicação é denominado *tópico* (*topic*). Com isso o desenvolvimento do *software* do robô pode ser criado de maneira distribuída entre diferentes componentes, de diferentes pacotes e bibliotecas, conectados através de um mesmo tópico. Caso seja necessário o envelopamento de informações entre diferentes nós, ou a criação de uma nova função, basta-se para isso que se crie um novo nó, e o conecte o mesmo através de um *tópico*.

O modelo de comunicação utilizado pela ROS é baseado no *Publisher-Subscriber*. Nesse modelo, um *Subscriber* requer uma mensagem do *Publisher* com informações, por exemplo, dos sensores, câmeras, etc. O *Publisher* publica para todos os *Subscribers* que estão aguardando a sua mensagem através de um canal em comum (o *tópico*). O *tópico* por sua vez armazena uma sequência de mensagens e as envia para os *nós Subscribers*. Dessa forma o fluxo de dados é gerado conforme o *Subscribers* geram pedidos de mensagens, sendo que os *Publishers* publicam essas mensagens no canal de comunicação *tópico*.

Esse modelo de comunicação é assíncrono, e apresenta algumas vantagens devido a alta dinâmica de configuração de conexão entre os nós desta rede. Isso acontece porque os nós da rede do ROS podem ser criados e destruídos a qualquer momento e os nós *Publishers* não possuem qualquer conhecimento sobre os nós *Subscribers*.

Na Figura 2.8 é apresentado um esquemático de exemplo visual de como funciona a comunicação entre os diferentes elementos de rede do ROS.

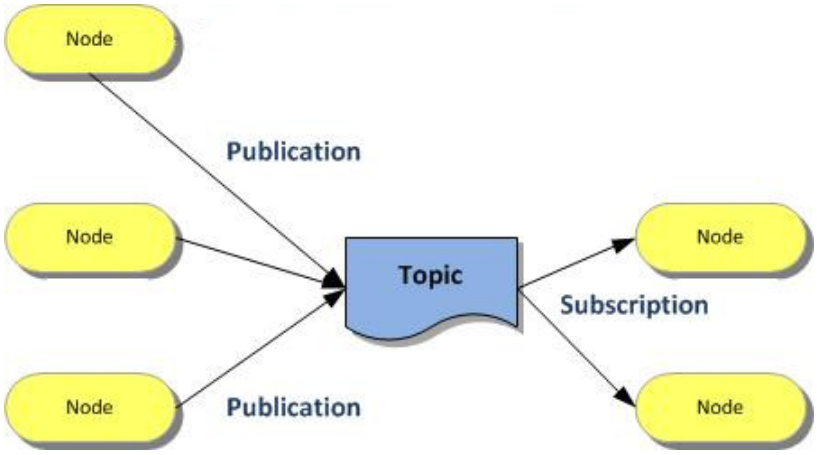

Figura 2.8: Esquemático que exemplifica graficamente o funcionamento da comunicação utilizada pela ROS.

2.3 Plataforma SoCKit Arrow Terasic com Altera FPGA Cyclone V

A placa de desenvolvimento da Arrow SoCKit [28] (vide Figura 2.9) possui como objetivo o aprendizado sobre o desenvolvimento de soluções usando sistemas embarcados baseados em arquitetura *ARM*, junto com o uso de FPGAs.

A placa combina uma processador Dual ARM Cortex-A9 com periféricos e interface de memória, um *Hard Processor System* (HPS) construido ao redor de uma placa FPGA Altera Cyclone V, com uma arquitetura de comunicação de alta velocidade.

O SoCKit (da forma em que é desenvolvido pela Terasic) implica em um sistema com baixa taxa de consumo energética e alta flexibilidade para o desenvolvimento de diferentes aplicações, com abordagens voltadas para o *software* e *hardware*.

A placa da SoCKit possui como principais características:

- FPGA Altera Cyclone V SoC 5CSXFC6D6F31 com 110 mil LE (Logical Elements)
- Dual ARM Cortex-A9 (HPS)
- 5140 Kbits de mémoria embarcada para configuração da FPGA
- 6 PLL (Phase-Locked Loop Basic)
- 1 GB DDR3 SDRAM para a FPGA
- 1 GB DDR3 SDRAM para o HPS
- 128 MB de memória flash para o HPS
- Possilidade de utilizar um cartão micro SD para o HPS
- Periféricos de entrada e saída (botões, leds, sensores, displays, usb)

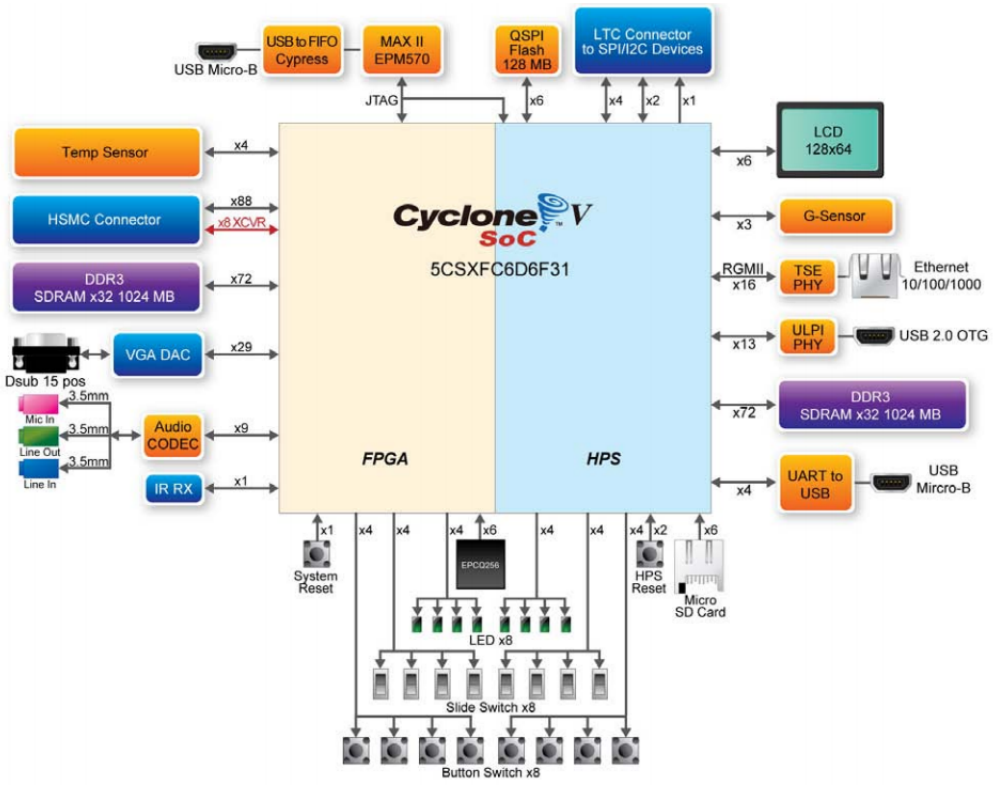

Figura 2.9: Diagrama com os componentes presentes no Kit de Desenvolvimento e as conexões desses periféricos

2.4 Modelagem Matemática

O modelo do quadrirrotor é descrito a partir de equações de um corpo rígido com seis graus de liberdade (6DOF), baseado no modelo descrito em [29] e [30].

2.5 Sistemas de Coordenadas

No modelo utilizado, existem quatro sistemas de coordenadas principais: (a) sistema de coordenas inercial global O_w, (b) o sistema de cordenadas do corpo rígido do quadrirrotor O_b, com a sua origem situada no centro de massa do AR.Drone, (c) o sistema de coordenadas da câmera frontal O_c, e (d) o sistema de coordenas bidimensional da imagem da câmera frontal O_i. No último, estarão representados os pixels presentes na imagem obtida pela câmera frontal do AR.Drone. Esses sistemas de coordenas estão presentes na Figura 2.10.

Figura 2.10: Representação dos diferentes sistemas de coordenadas.

2.6 O Sistema de Orientação

Para descrever a orientação do quadrirrotor em relação ao sistema de coordenadas inercial utilizou-se os ângulos de Euler (vide Figura 2.11).

O uso dos ângulos de Euler é mais intuítivo do que o uso de quaternions, que utilizam quatro parâmetros e possuem maior custo computacional. Porém a sua desvantagem (se comparado com o uso de quaternions) é a representação da orientação em determinadas movimentações que possuem singularidades.

Essas singularidades ocorrem quando as transformações de rotações de diferentes ângulos se alinham, e consequentemente a movimentação de um causa a movimentação do outro. Dessa forma há uma perda de um dos graus de liberdade.

Por exemplo, se o quadrirrotor atingir um ângulo de arfagem de 90°, utilizando a convenção da ordem de rotações em x-y-z, os ângulos de rolagem e guinada se tornam ambiguos. Porém, como o *software* do AR.Drone é programado para entrar em modo de emergência quando o mesmo atinge ângulos de rolagem e arfagem próximos de 90°. No nosso caso, é seguro afirmar que para a nossa aplicação essas singularidades nunca serão atingidas.

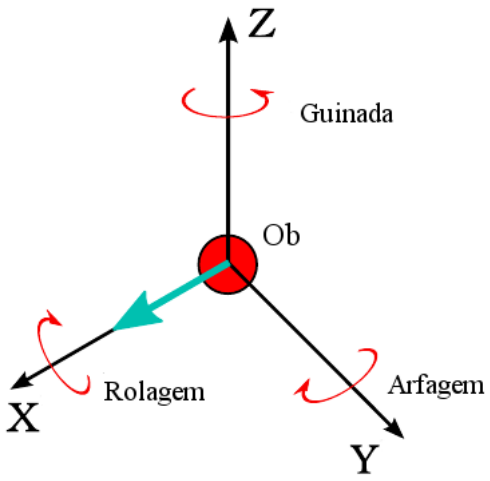

Figura 2.11: Representação do sistema de orientação utilizado rolagem(ϕ) - arfagem(θ) - guinada(ψ).

A convenção utilizada para as rotações entre os diferentes sistemas de coordenadas é a transformação de rotação na ordem rolagem-arfagem-guinada. Por exemplo, para representar rotações entre os sitemas de coordenadas O_b e O_w utilizamos a matriz de rotação presente em 2.1, 2.2 e 2.3.

$$R_{bw} = R_{bw}(\psi) \times R_{bw}(\theta) \times R_{bw}(\phi) \tag{2.1}$$

$$R_{bw} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi + C_\phi S_\psi & -C_\phi S_\theta C_\psi + S_\phi S_\psi \\ -C_\theta S_\psi & -S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi + S_\phi C_\psi \\ -S_\theta & -S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} C_\phi \\ C_\theta \\ C_\psi \\ S_\phi \\ S_\theta \\ S_\psi \end{bmatrix} = \begin{bmatrix} cos(\phi) \\ cos(\theta) \\ cos(\psi) \\ sen(\phi) \\ sen(\theta) \\ sen(\psi) \end{bmatrix} \tag{2.3}$$

2.7 Modelagem Matemática do AR.Drone

Quadrirrotores possuem a característica de serem sistemas sub-atuados, ou seja, apesar de poderem se movimentar com 6DOF o sistema é atuado apenas por quatro atuadores, os seus propulsores. Isso implica que para algumas movimentações alguns dos seus graus de liberdade não se comportam independentes um do outro. Isso acontece, por exemplo, se alterarmos o ângulo de rolagem o quadrirrotor irá se movimentar lateralmente, porque os propulsores não alteram o seu

ângulo de ataque e as forças de empuxo que sustentam o quadrirrotor sempre estão sendo aplicadas verticalmente em O_b.

A Figura 2.12 possui um esquemático das forças causadas pelos propulsores que ilustram esse comportamento.

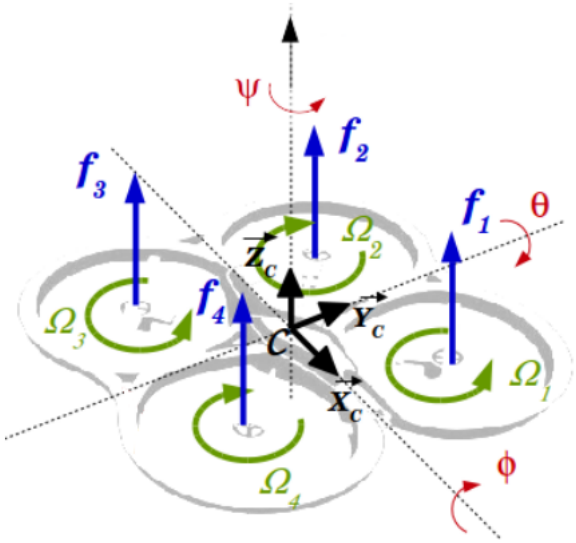

Figura 2.12: Representação de um quadrirrotor com as forças de empuxo f_i (*thrust*) causadas pelas hélices dos rotores girando com velocidades Ω_i com $i = [1, 4]$ representando cada atuador. Há também as indicações dos ângulos de guinada ψ, rolagem ϕ e arfagem θ.

Para simplificar a modelagem, assumiu-se as seguintes características para o sistema:

- O AR.drone é um corpo rígido e o seu chassi não se deforma com as forças aplicadas.
- As forças de empuxo causados pelos propulsores são lineares, ou seja, a relação entre força de empuxo e velocidade de giro do motor sempre se comporta com uma taxa constante independentemente de outros fatores.
- As dinâmicas internas do quadrirrotor são constantes. Isto é, apesar da bateria descarregar e os propulsores responderem com menor força, considera-se que esse efeito não acontece.

Baseado nos modelos presente em [29] e [30] obtem-se as dinâmicas translacionais e rotacionais do AR.Drone, considerando que as forças presentes são: (a) força gravitacional, (b) força de arrasto e (c) força de empuxo.

A força gravitacional (vide equação 2.4) é negativa na direção do eixo-z do sistema de coordenadas global, com m sendo a massa do VANT e g o módulo da aceleração da gravidade.

$$f_g = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{2.4}$$

A força de arrasto (vide equação 2.5) atua na posição oposta do vetor velocidade linear do AR.Drone, e foi simplificada para atuar sobre o corpo rígido do quadrirrotor. Como as forças de arrasto em fluídos são proporcionais ao módulo da velocidade do corpo dentro do fluído temos que os parâmetros $[A_x(\dot{x}), A_y(\dot{y}), A_z(\dot{z})]$, que representam pertubações causadas pelas forças de arrasto, também são dependentes dos módulos das velocidades lineares do quadrirrotor.

$$f_d = \begin{bmatrix} A_x(\dot{x}) \\ A_y(\dot{y}) \\ A_z(\dot{z}) \end{bmatrix} \tag{2.5}$$

A força de empuxo (vide equação 2.6) atua na direção positiva do eixo-z do sistema de coordenadas do AR.Drone (O_b). Porém, a mesma possui componentes nos outros eixos, dependendo da orientação do AR.Drone, representado pelos seus ângulos de Euler. Os componentes das forças em cada eixo são $[F_{t,x}, F_{t,y}, F_{t,z}]$.

$$f_t = \begin{bmatrix} F_{t,x}(-cos(\phi)sen(\theta)cos(\psi) + sen(\phi)sen(\psi)) \\ F_{t,y}(cos(\phi)sen(\theta)sen(\psi) + sen(\phi)cos(\psi)) \\ F_{t,z}(cos(\phi)cos(\theta)) \end{bmatrix} \tag{2.6}$$

O sistema dinâmico do AR.Drone é representado pela equação 2.7, sendo as suas variáveis definidas na Tabela 2.1.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{\eta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{F_{t,x}(-cos(\phi)sen(\theta)cos(\psi)+sen(\phi)sen(\psi))+A_x(\dot{x})}{m} \\ \frac{F_{t,y}(cos(\phi)sen(\theta)sen(\psi)+sen(\phi)cos(\psi))+A_y(\dot{y})}{m} \\ -g + \frac{F_{t,z}(cos(\phi)cos(\theta))+A_z(\dot{z})}{m} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ M(\eta)^T(\tau(\eta) - C(\eta, \dot{(\eta)})\dot{(\eta)}) \end{bmatrix} \tag{2.7}$$

Variável	Definição
$\dot{x}, \dot{y}, \dot{z}$	Velocidades do corpo rígido.
$\ddot{x}, \ddot{y}, \ddot{z}$	Acelerações do corpo rígido.
ϕ, θ, ψ	ângulos de rolagem, arfagem e guinada.
$\dot{\phi}, \dot{\theta}, \dot{\psi}$	Velocidades ângulares do VANT ao redor dos seus eixos.
$\ddot{\eta}$	Aceleração ângular do VANT ao redor dos seus eixos.
$F_{t,x}, F_{t,y}, F_{t,z}$	Forças de empuxo causadas pelos rotores.
m	Massa do AR.Drone.
g	Módulo da aceleração da gravidade.
$M(\eta)^T$	Matriz transposta da primeira derivada no tempo da posição ângular do VANT multiplicada pelos momentos de inércia em cada eixo.
τ_η	Torques sobre o quadricoptero causados pelos rotores.
$C(\eta, \dot{(\eta)})$	Efeito Coriolis sobre o VANT

Tabela 2.1: Variáveis utilizadas no modelo do AR.Drone.

2.7.1 Simplificação do Modelo do AR.Drone

O modelo descrito na equação 2.7 possui natureza completamente não-linear, e como será utilizado o controlador embarcado desenvolvido pelo fabricante do AR.Drone [7] é possível assumir alguns parâmetros para simplificar esse modelo e obter um modelo em espaço de estados com entradas de controle similares as do controlador embarcado do AR.Drone.

O controlador do AR.Drone utiliza como entradas de referência os ângulos de rolagem e arfagem, a velocidade ângular do ângulo de guinada e a velocidade vertical do AR.Drone, conforme 2.8.

$$\boldsymbol{u}(t) = \begin{bmatrix} \phi_{ref} & \theta_{ref} & \dot{\psi}_{ref} & \dot{z}_{ref} \end{bmatrix}^T \tag{2.8}$$

Com isso, baseando-se no modelo descrito em [31], e assumindo que o vetor de espaço de estados utilizado pelo AR.Drone será conforme a equação 2.9.

$$\boldsymbol{X}(t) = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \phi & \theta & \psi & \dot{\psi} \end{bmatrix}^T \tag{2.9}$$

O modelo simplificado (presente em 2.10) assume que as dinâmicas dos ângulos de rolagem e arfagem são estabilizadas pelo controle de malha fechada implementado no *software* embarcado no AR.Drone, e que os comandos do controlador são enviados em um curto intervalo de tempo (δ_t).

$$\boldsymbol{X}(t+1) = \begin{bmatrix} x(t+1) \\ y(t+1) \\ z(t+1) \\ \dot{x}(t+1) \\ \dot{y}(t+1) \\ \dot{z}(t+1) \\ \phi(t+1) \\ \theta(t+1) \\ \psi(t+1) \\ \dot{\psi}(t+1) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \\ \phi(t) \\ \theta(t) \\ \psi(t) \\ \dot{\psi}(t) \end{bmatrix} + \delta_t \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \\ \ddot{x}(\boldsymbol{X}(t)) \\ \ddot{y}(\boldsymbol{X}(t)) \\ \ddot{z}(\boldsymbol{X}(t), \boldsymbol{u}(t)) \\ \dot{\phi}(\boldsymbol{X}(t), \boldsymbol{u}(t)) \\ \dot{\theta}(\boldsymbol{X}(t), \boldsymbol{u}(t)) \\ \dot{\psi}(t) \\ \ddot{\psi}(\boldsymbol{X}(t), \boldsymbol{u}(t)) \end{bmatrix} \tag{2.10}$$

Semelhante ao modelo presente na equação 2.7 temos as equações 2.11 e 2.12 para a dinâmica no *plano-xy* horizontal, em função das entradas do controlador.

$$\ddot{x}(\boldsymbol{X}(t)) = k_1(cos(\theta(t))cos(\psi(t))sen(\theta(t)) - sen(\theta)sen(\psi)) - k_2\dot{x}(t) \tag{2.11}$$

$$\ddot{y}(\boldsymbol{X}(t)) = k_1(-cos(\theta(t))sen(\psi(t))sen(\phi(t)) - sen(\theta)cos(\psi)) - k_2\dot{y}(t) \tag{2.12}$$

As equações presentes em 2.13, 2.14, 2.15 e 2.16 descrevem a influência linear das variáveis de controle sobre o estado de espaços.

$$\dot{\phi}(\boldsymbol{X}(t), \boldsymbol{u}(t)) = k_3 u_\phi(t) - k_4\phi(t) \tag{2.13}$$

$$\dot{\theta}(\boldsymbol{X}(t), \boldsymbol{u}(t)) = k_3 u_\theta(t) - k_4\theta(t) \tag{2.14}$$

$$\ddot{\psi}(\boldsymbol{X}(t), \boldsymbol{u}(t)) = k_5 u_\psi(t) - k_6\dot{\psi}(t) \tag{2.15}$$

$$\ddot{z}(\boldsymbol{X}(t), \boldsymbol{u}(t)) = k_7 u_{\dot{z}}(t) - k_8\dot{z} \tag{2.16}$$

Os parâmetros k_1 à k_8 foram estimados através dos dados coletados por diversos voos realizados por [31].

2.8 Modelo da Câmera Frontal e Projeção de Imagens

Uma câmera funciona mapeando objetos presentes em um sistema de coordenadas tridimensional para um sistema de coordenadas bidimensionais (uma imagem). Esse processo é representado nas Figuras 2.13 e 2.14.

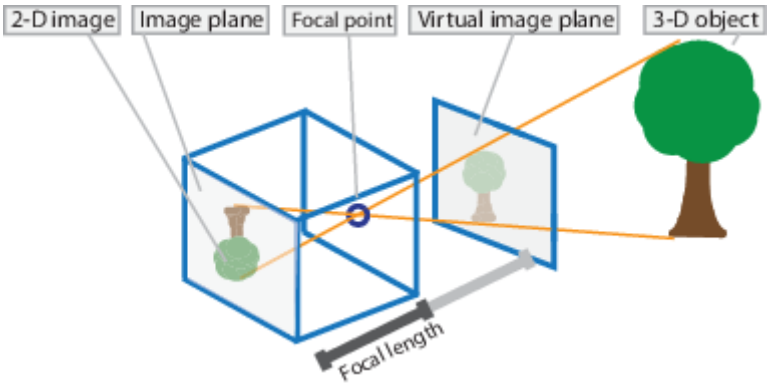

Figura 2.13: Funcionamento de uma câmera *pinhole* [8].

A câmera frontal presente no AR.Drone é modelada através do modelo de projeção de câmera *pinhole* [32].

Nesse modelo um ponto em O_w com distância relativa à câmera representado por $p_c = [x \;\; y \;\; z]^T$ é projetado no plano da imagem na posição $[u \;\; v]^T$ pertencente à O_i. Essa projeção é representada pelas equações 2.17, 2.18 e 2.19.

$$\lambda \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} C_x \\ C_y \end{bmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \frac{r'}{r} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} \tag{2.17}$$

$$r = \sqrt{\frac{x^2 + y^2}{z^2}} \tag{2.18}$$

$$r' = \frac{1}{\omega} arctan\left(2 \;\; r \;\; tan(\frac{\omega}{2})\right) \tag{2.19}$$

Na equação 2.17, os parâmetros intrísecos da câmera são: (a) distância focal da câmera ($f_x \;\; f_y$), (b) posição do pixel central na imagem ($C_x \;\; C_y$) e (c) a distorção radial ω. Os parâmetros extrinsicos são a orientação da câmera em relação ao sistema de coordenadas global O_w (que é obtido através da orientação do VANT) e a posição da câmera. As características intrisecas da câmera são obtidas através da calibração da mesma.

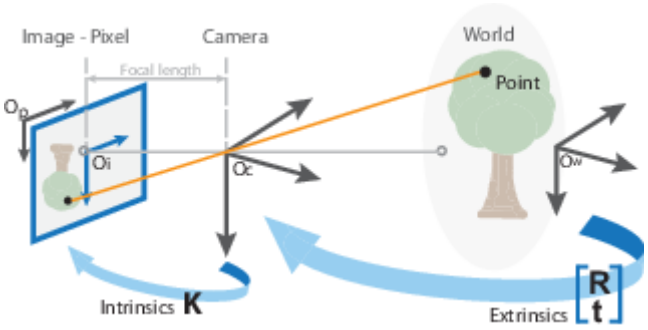

Figura 2.14: Projeção de ponto presente em O_w para plano da imagem O_i [8].

2.9 Filtro de Kalman

O filtro de Kalman [33] é um método iterativo de estimação estocástica de estados de um sistema dinâmico linear, que utiliza as medições de observações ao decorrer do tempo, e cuja evolução possui o objetivo de predizer os estados futuros do sistema, ao mesmo tempo em que os erros (incertezas) de estimação são minimizados.

O modelo do sistema dinâmico linear em tempo discreto é representado pelas equações 2.20 e 2.21.

$$\boldsymbol{x}_k = \boldsymbol{A}_k \cdot \boldsymbol{X}_{k-1} + \boldsymbol{B}_k \cdot \boldsymbol{u}_k + \boldsymbol{w}_k \tag{2.20}$$

A equação 2.20 descreve o modelo da evolução do processo com o passar do tempo.

$$\boldsymbol{y}_k = \boldsymbol{H}_k \cdot \boldsymbol{X}_k + \boldsymbol{v}_k \tag{2.21}$$

A equação 2.21 descreve o modelo das medições.

Os vetores $\boldsymbol{x}_k$, $\boldsymbol{u}_k$ e $\boldsymbol{y}_k$ representam no instante de tempo k, respectivamente, o vetor de estados do sistema, o vetor de controle e o vetor de medições. As matrizes $\boldsymbol{A}_k$, $\boldsymbol{B}_k$ e $\boldsymbol{H}$ representam no instante de tempo k, respectivamente, o modelo de trasição de estados (modelo da planta ou processo) e o modelo de observação do sistema (modelo de medição). Os vetores $\boldsymbol{w}_k$ e $\boldsymbol{v}_k$ representam os ruídos do processo e da observação, respectivamente.

O estado do filtro de Kalman é representado por duas variáveis: (a) a estimativa do vetor de estados $\hat{\boldsymbol{x}}_{k|k}$ para o próximo estado dado as informações obtidas no estado atual, e (b) a matriz de covariância do erro de estimação $\boldsymbol{P}_{k|k}$ do próximo estado dado o estado atual.

O método de filtragem do filtro de Kalman consiste em duas etapas: (a) a predição e (b) a correção 2.15. Neste contexto, a etapa de predição utiliza os dados obtidos no estado anterior para reaizar uma estimativa do estado atual do sistema. A etapa de correção realiza a predição com o auxílio das observações atuais mensuradas para corrigir e melhorar a estimativa do estado atual.

As equações que descrevem a etapa de predição são representadas nas equações 2.22 e 2.23.

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{A}_{k-1} \cdot \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}_k \cdot \boldsymbol{u}_k \tag{2.22}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{A}_{k-1} \cdot \boldsymbol{P}_{k-1} \cdot \boldsymbol{A}_{k-1}^T + \boldsymbol{Q}_k \tag{2.23}$$

As equações que descrevem a etapa de correção são representadas nas equações 2.24, 2.25 e 2.26. Neste caso, K representa o ganho de Kalman e esse fator conforme é atualizado é responsável por gerar estimativas do mínimo erro quadrático.

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \cdot \boldsymbol{H}_k^T \cdot \left(\boldsymbol{H}_k \cdot \boldsymbol{P}_{k|k-1} \cdot \boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1} \tag{2.24}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \cdot \left(\boldsymbol{y}_k - \boldsymbol{H}_k \cdot \hat{\boldsymbol{x}}_{k|k-1}\right) \tag{2.25}$$

$$\boldsymbol{P}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k \cdot \boldsymbol{H}_k\right)^T + \boldsymbol{K}_k \cdot \boldsymbol{R}_k \cdot \boldsymbol{K}_k^T \tag{2.26}$$

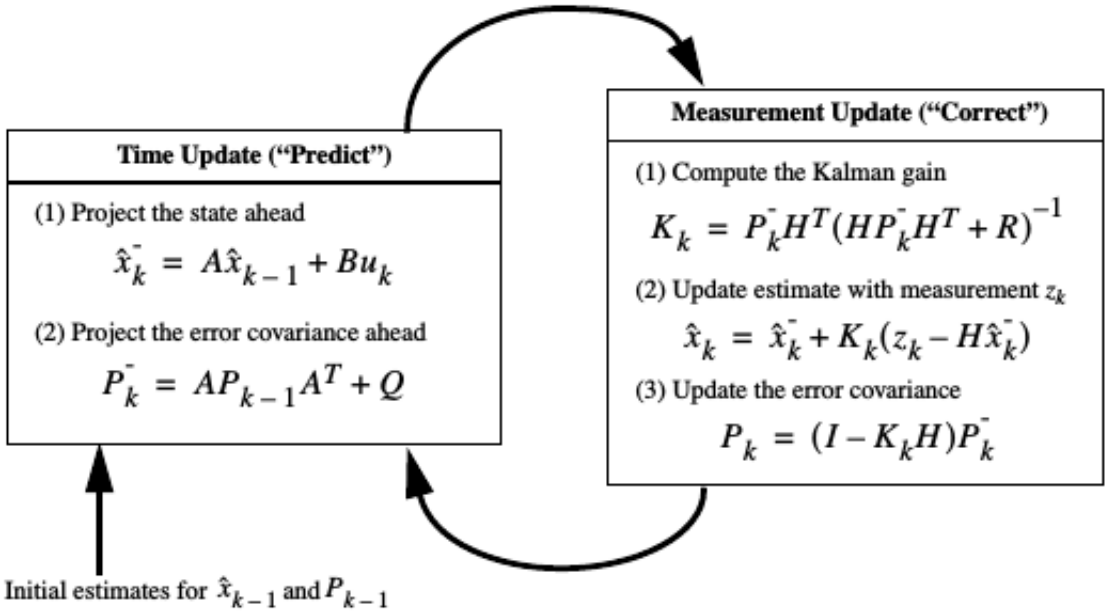

Figura 2.15: A estrutura completa do filtro de Kalman representando as suas duas etapas sendo executadas ciclicamente. [9]

2.9.1 Filtro de Kalman Estendido (EKF)

Conforme apresentado em 2.9, o filtro de Kalman se baseia em modelos lineares para realizar a predição dos estados do sistema e a correção dos estados estimados. Porém, o sistema do quadrirrotor como apresentado na seção 2.7 possui características não-lineares e o uso de um filtro de Kalman simples pode não ser robusto o bastante para o nosso sistema.

O filtro de Kalman estendido (EKF) (representado na Figura 2.16) é uma versão do filtro de Kalman capaz de lidar com funções não-lineares para realizar as estimativas do estado do sistema. O princípio aplicado ao EKF é a aplicação de uma linearização de primeira ordem do modelo não-linear do sistema através da expansão da série de Taylor de primeira ordem ao redor da estimativa atual.

As modificações entre os dois filtros na representação do modelo do sistema dinâmico é a aplicação de uma função não-linear $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ e por uma função $h(\boldsymbol{x}_k)$, substituindo as matrizes A e B na descrição dos estados e substituindo a matriz H na equação para obtenção das medidas do sistema, respectivamente. O novo modelo é descrito nas equações 2.27 e 2.28.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{2.27}$$

$$\boldsymbol{y}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{2.28}$$

Na etapa de predição e correção o EKF utiliza Jacobianos da função f e h a fim de substituir as matrizes lineares A e H do filtro de Kalman convencional. Os Jacobianos estão representados nas equações 2.29 e 2.30 em relação à um estado $\boldsymbol{x}$.

$$\boldsymbol{A}_{\boldsymbol{x}} = \left.\frac{\partial f(\boldsymbol{x}_k, \boldsymbol{u}_k)}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{2.29}$$

$$\boldsymbol{H}_{\boldsymbol{x}} = \left.\frac{\partial h(\boldsymbol{x}_k)}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k} \tag{2.30}$$

Com essas modificações a etapa de predição é representada nas equações 2.31 e 2.32.

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{2.31}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{A}_{\boldsymbol{x}_{k-1}} \cdot \boldsymbol{P}_{k-1} \cdot \boldsymbol{A}^T_{\boldsymbol{x}_{k-1}} + \boldsymbol{Q}_k \tag{2.32}$$

A etapa de predição do EKF é representada pelas equações 2.33, 2.34 e 2.35.

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \cdot \boldsymbol{H}^T_{\boldsymbol{x}_k} \cdot \left(\boldsymbol{H}_{\boldsymbol{x}_k} \cdot \boldsymbol{P}_{k|k-1} \cdot \boldsymbol{H}^T_{\boldsymbol{x}_k} + \boldsymbol{R}_k\right)^{-1} \tag{2.33}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \cdot \left(\boldsymbol{y}_k - \boldsymbol{H}_{\boldsymbol{x}_k} \cdot \hat{\boldsymbol{x}}_{k|k-1}\right) \tag{2.34}$$

$$\boldsymbol{P}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k \cdot \boldsymbol{H}_{\boldsymbol{x}_k}\right)^T + \boldsymbol{K}_k \cdot \boldsymbol{R}_k \cdot \boldsymbol{K}^T_k \tag{2.35}$$

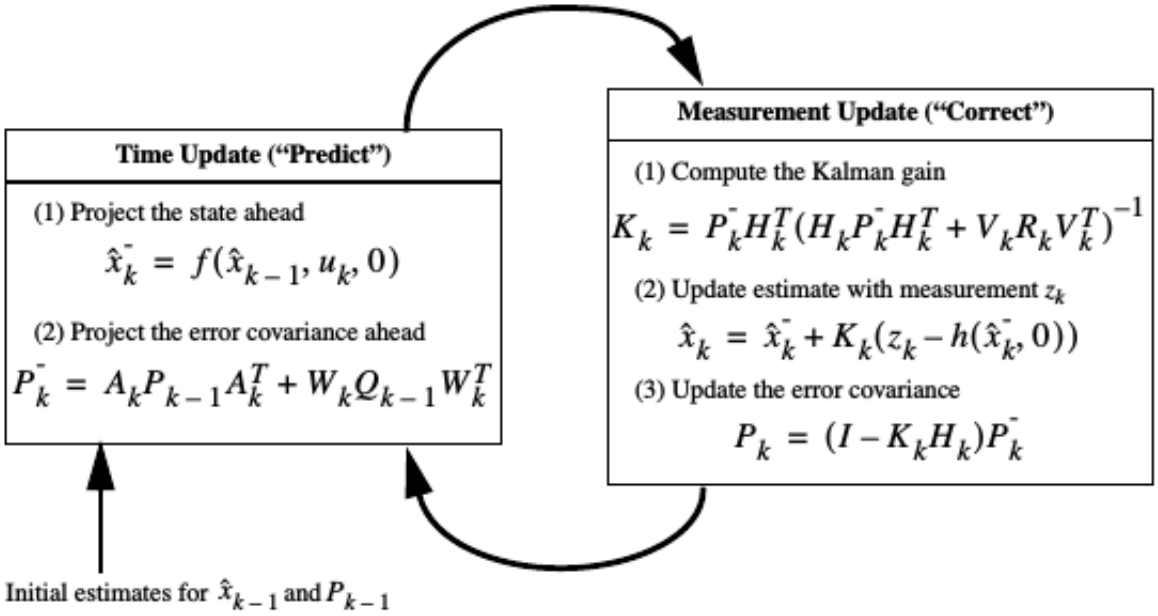

Figura 2.16: A estrutura completa do filtro de Kalman estendido representando as suas duas etapas sendo executadas ciclicamente. [9]

Capítulo 3

Desenvolvimento

Esse capítulo relata as etapas de desenvolvimento e a integração das diversas ferramentas utilizadas, além de descrever a arquitetura da implementação do SLAM Monocular no AR.Drone.

3.1 Integração do Harware aliado a plataforma ROS e Gazebo

Como descrito no Capítulo 2, o VANT utilizado transmite e recebe dados pela rede Wi-Fi ad-hoc para um computador remoto. O computador de base (executando o *framework* ROS) cria diferentes tópicos específicos para a comunicação entre o AR.Drone e os nós implementados para processar os dados do VANT. Os subtópicos a seguir discorrem sobre os diferentes nós e tópicos utilizados pelos diversos pacotes utilizados com o ROS.

3.1.1 Pacote do AR.Drone Driver

Esse driver implementa um *nó* que lê e escreve os dados pelas portas UDP nos canais de comunicação, conforme explicado em 2.1.4. Os *tópicos* criados pelo ardrone_autonomy. Os mesmos são */navdata*, */front/image_raw* e o */cmd_vel* responsáveis por: (a) possuir um stream de dados lidos pelos sensores do VANT, conforme a Tabela 3.1, (b) possuir um stream de vídeo da imagem da câmera frontal, e (c) receber os comandos de controle para a movimentação do AR.Drone.

Nome no tópico	Definição	Unidade de medida
header	Cabeçalho da mensagem possuindo um contador (*stamp*) e o id do tópico	*
batteryPercentage	Nível da bateria do VANT	[°]
state	Estado do VANT, i.e. em solo, decolando, pousando, pairando e emergência	*
rotX, rotY, rotZ	Rotações em torno dos eixos x, y e z	[°]
altd	Altitude estimada do VANT	[mm]
vx, vy, vz	Estimativa da velocidade linear nos eixos x, y e z	[mm/s]
ax, ay, az	Aceleração linear no eixos x, y e z	[g]
magX, magY, magZ	Leitura do magnetômetro em torno dos eixos x, y e z	*
pressure	Pressão medida pelo barômetro	*
temp	Temperatura	*
$\text{wind}_{s}peed$	Velocidade Estimada do Vento	*
$\text{wind}_{a}ngle$	Ângulo Estimado do Vento	*
$\text{wind}_{c}omp_{a}ngle$	Compensação estimada para o vento	*
motor1, motor2, motor3, motor4	Velocidade dos motores em PWM	*
tm	Medição do Tempo - *timestamp*	[ms]

$*$ - Não se aplica, ou não está implementado ou não é utilizado.

Tabela 3.1: Mensagens presentes no tópico */navdata*.

Desse conjunto de dados utilizaremos as velocidades $(\dot{x}, \dot{y}, \dot{z})$, acelerações $(\ddot{x}, \ddot{y}, \ddot{z})$, a orientação (ϕ, θ, ψ) e a altitude z estimadas pelo VANT.

3.1.2 Simulação do Quadrirrotor utilizando o Gazebo

As mesmas características que tornam o AR.Drone uma ferramenta excepcional para a robótica, por ser um VANT de baixo custo, também o tornam um equipamento bastante sensível à danos causados por quedas e ou manipulação indevida.

Infelizmente, algumas das abordagens planejadas para serem desenvolvidas nesse trabalho encontraram a barreira física do uso do AR.Drone disponível.

No contexto deste trabalho, o AR.Drone 1.0 utilizado faz parte dos equipamentos do laboratório (LEIA-GRACO) por cerca de 2 anos e já foi utilizado por dois projetos de trabalho de graduação anteriores.

Conforme o *hardware* foi sendo utilizado aconteceram eventuais quedas e até pousos de emergência, que danificaram e/ou empenaram um pouco as hélices (que são bastantes sensíveis) ou os motores dos propulsores.

Após um determinado ponto, a utilização correta do drone se torna bastante prejudicada. No caso desse trabalho, apesar de desempenar as hélices e troca-las, conforme descrito no tutorial diponível pelo fabricante [34], o drone continuou com problemas para decolar ou manter-se no estado pairando para a obtenção dos dados de navegação.

A abordagem para contornar esse problema foi a utilização do Gazebo, que é um dos pacotes de simulação compatíveis com a ROS, para obter uma simulação do quadrirrotor e de uma possível cena em que o AR.Drone possa navegar.

O Gazebo [35] é uma plataforma *opensource* de simulação criada para implementação e testes em sistemas robóticos. A mesma possui interface gráfica de fácil entendimento e também possui a capacidade de ser expandida através da utilização de *plugins* e pacotes. Um dos *plugins* que podem ser instalados com o Gazebo é o *gazebo_ros*, que é uma interface entre o ROS e o Gazebo.

Para simular o quadrirrotor foi utilizado um pacote de simulação do AR.Drone [36], baseado no modelo de simulação de quadrirrotores implementado em [37].

As vantagens desse pacote de simulação é que (em sua versão atual) os topicos criados por ele tentam ser o mais semelhantes possíveis com os tópicos criados pelo *ardrone_driver*. Isto é, para a aplicação não há diferenças entre ser chamada para atuar sobre um AR.Drone real em voo ou o pacote de simulação junto com o Gazebo.

As Figuras 3.1 e 3.2 ilustram uma aplicação de exemplo para o controle do AR.drone utilizando um controle *joystick*, mostrando que para o nó do ROS *joy_node* não há mudanças nos seus tópicos para manipular um AR.drone real ou presente na simulação.

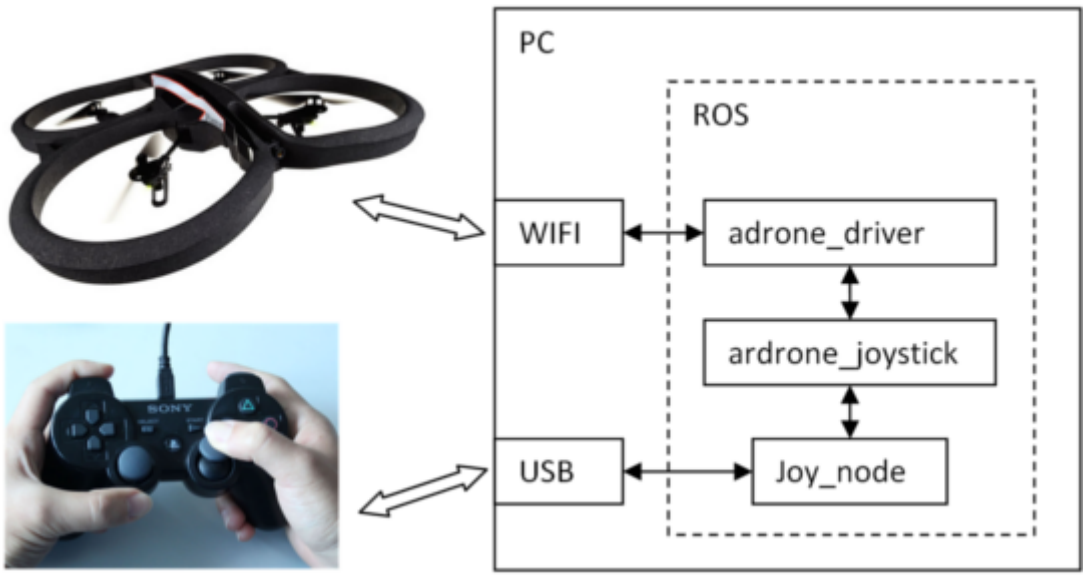

Figura 3.1: AR.Drone sendo controlado via joystick utilizando o *ardrone_ autonomy*.

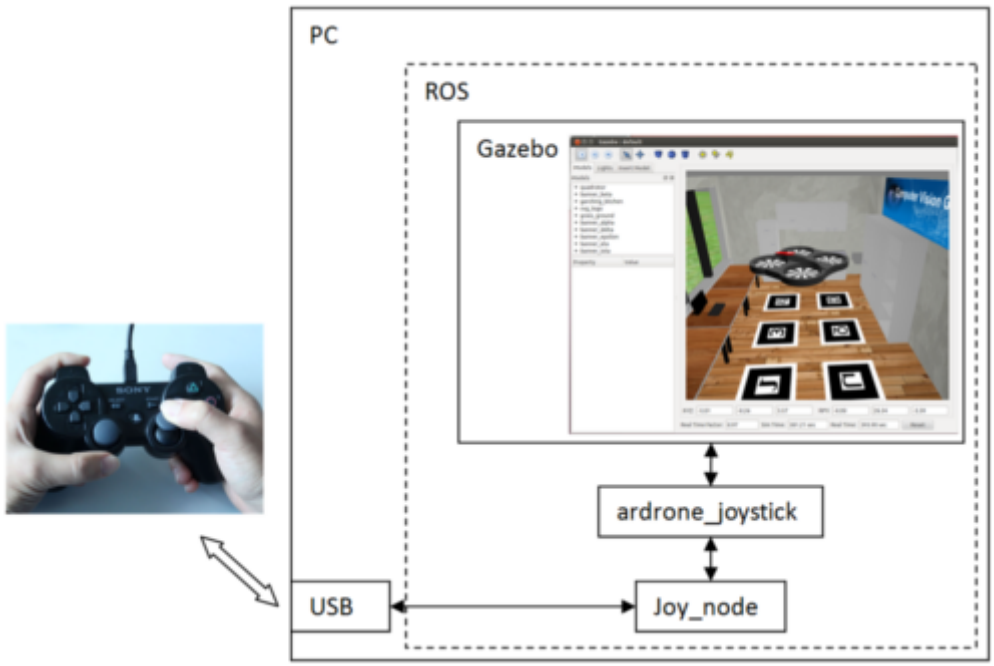

Figura 3.2: AR.Drone dentro da simulação do Gazebo sendo controlado via joystick utilizando os mesmos tópicos do ROS.

As figuras 3.3 e 3.4 apresentam a visualização de uma simulação do Gazebo com o pacote de simulação do AR.Drone. Nas figuras é possível visualizar o ambiente de simulação que foi preenchido com diversos objetos, para que o algoritmo de localização visual funcione, assim como visualizar o modelo tridimensional do AR.Drone.

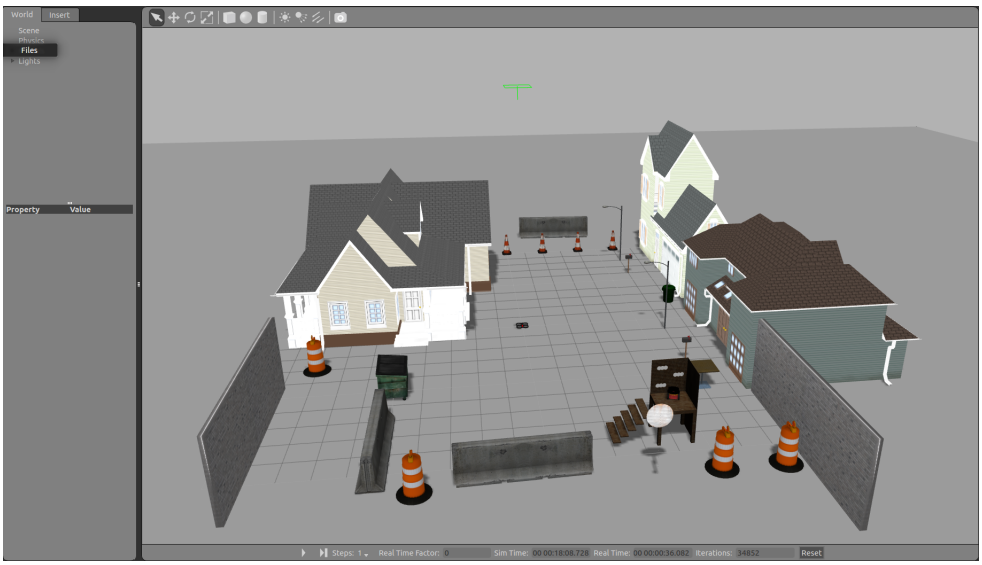

Figura 3.3: Visualização do ambiente de simulação do Gazebo com o modelo do ar.drone.

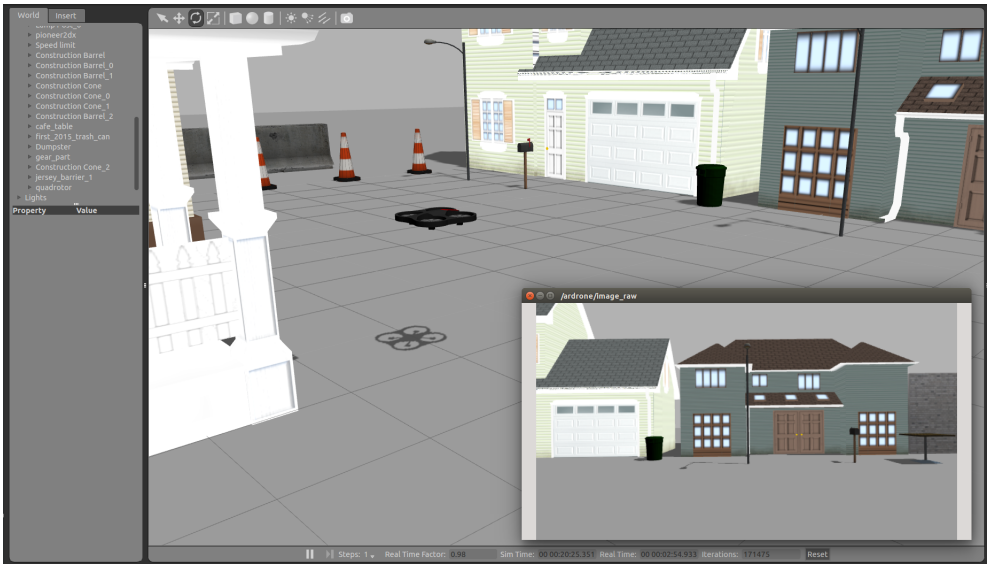

Figura 3.4: AR.Drone em voo no ambiente de simulação e a visualização da imagem obtida da câmera frontal.

3.1.3 Interface com o usuário

Para simplificar o uso da plataforma, assim como para se ter um maior controle sobre os parâmetros do nó do AR.Drone (seja utilizando a simulação com o Gazebo ou utilizando o ardrone_autonomy) foi implementada uma interface gráfica simples em C++ utilizando o Qt, que é um pacote de desenvolvimento de interfaces gráficas que utiliza o C++.

Com essa janela é possível: (a) enviar comandos diretamente para o AR.Drone, utilizando a sintáxe do ardrone_autonomy, (b) observar os seu parâmetros de comunicação, (c) observar os seus parâmetros de controle, o nó de estimação dos estados do AR.Drone, e (d) escolher como

controlar a posição do VANT, seja automáticamente com comandos ou utilizando o teclado.

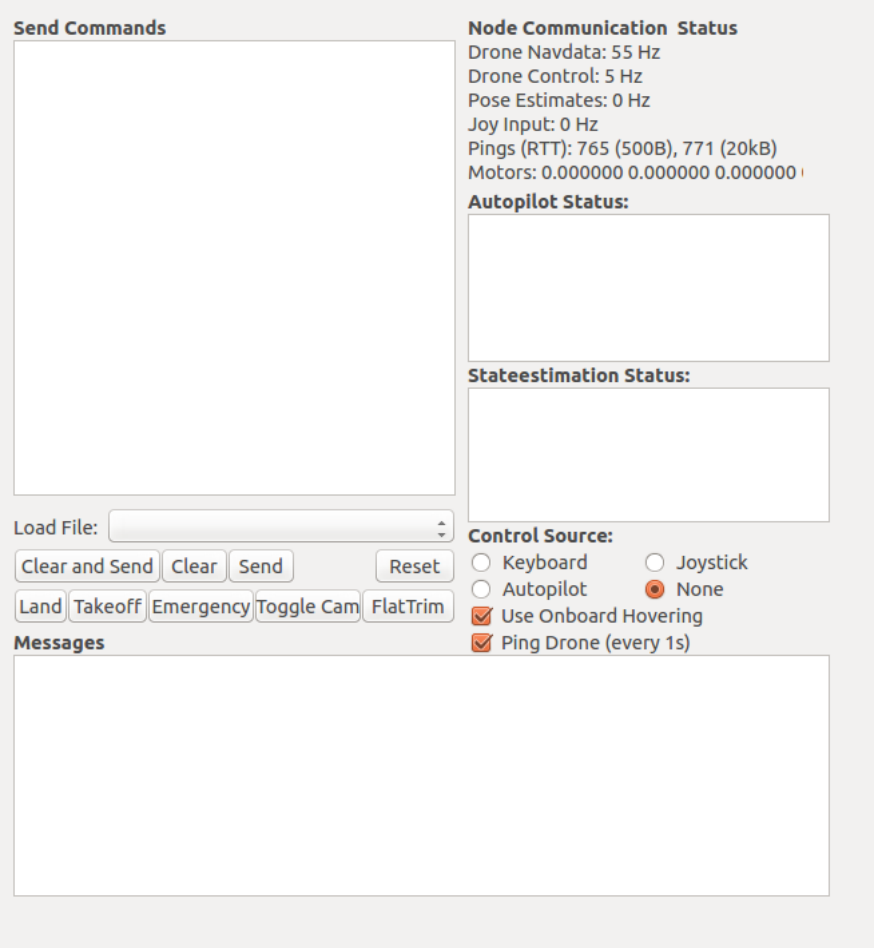

Figura 3.5: Interface Gráfica utilizada para ter informações sobre o AR.Drone e também publicar comandos nos seus tópicos de controle.

Na Figura 3.7 é possível visualizar a conexão entre os nós do ardrone_driver e o drone_gui, utilizando os diversos tópicos de comunicação usados pelo ardrone_driver. Adicionalmente, na Figura 3.7 é possível comparar (para fins de verificação) a mesma conexão, só que utilizando o Gazebo.

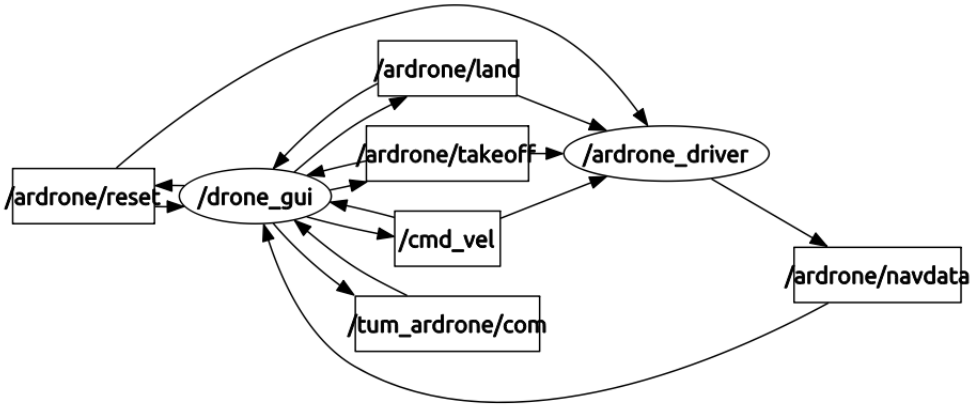

Figura 3.6: Representação gráfica das conexões utilizando tópicos (retangulos) entre os nós (elipses) do ardrone_driver e drone_gui (interface gráfica).

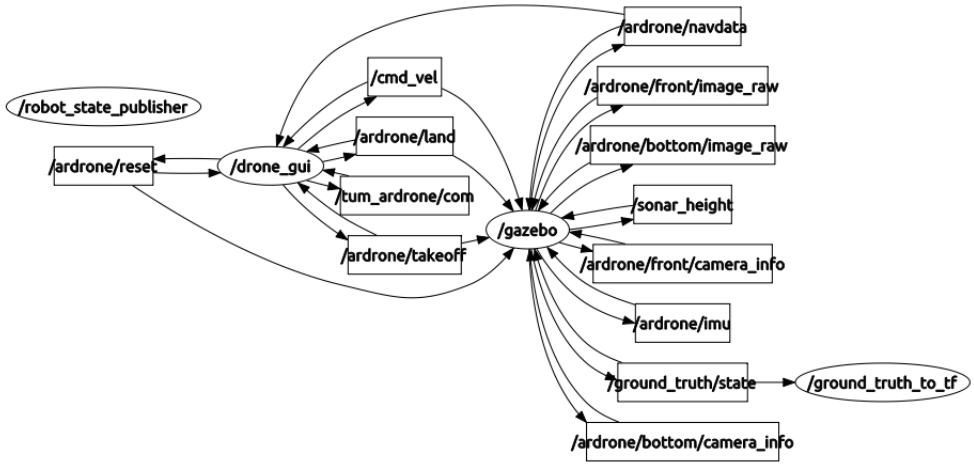

Figura 3.7: Representação gráfica das conexões utilizando tópicos (retangulos) entre os nós (elipses) do gazebo com plugins e do drone_gui (interface gráfica).

3.2 Calibração da Câmera Frontal

Com a drástica redução dos custos envolvidos na produção que tornam os preços acessíveis, nos últimos anos houve uma rápida popularização das câmeras digitais. Atualmente, devido a sua miniaturização, as câmeras estão presentes em *smartphones*, *tablets* e *webcams*. Porém, inerente aos seus parâmetros aos princípios físicos utilizados para obtenção das imagens, conforme 2, as câmeras possuem distorções, que em aplicações de odometria tornam a obtenção da projeção da imagem da câmera bastante discrepante aos dados esperados com uma câmera ideal.

A calibração da câmera é a obtenção de seus parâmetros intrísecos para serem utilizados na

correção das distorções. Para realizar a calibração utilizou-se o pacote do ROS *camera_ calibrator* que é baseado em técnicas de calibração de câmeras monoculares desenvolvidas por [38].

Para realizar a calibração é necessário obter diversas imagens de um objeto com padrões de fácil detecção e de dimensões conhecidas em diversas orientações e posições relativas à câmera. Foi utilizado a imagem de um tabuleiro de xadrez (*chessboard*) impresso em uma folha A3, conforme a figura 3.8.

Figura 3.8: Imagem da folha A3 impressa com um padrão de xadrez com 9x7 casas e 3.5 cm cada uma delas.

Para este caso, a ferramenta do ROS é bastante intuítiva, e conforme as imagens são sendo obtidas a ferramenta mostra a qualidade que a calibração vai possuir, conforme as imagens são adicionadas. A interface da ferramenta é mostrada na Figura 3.9.

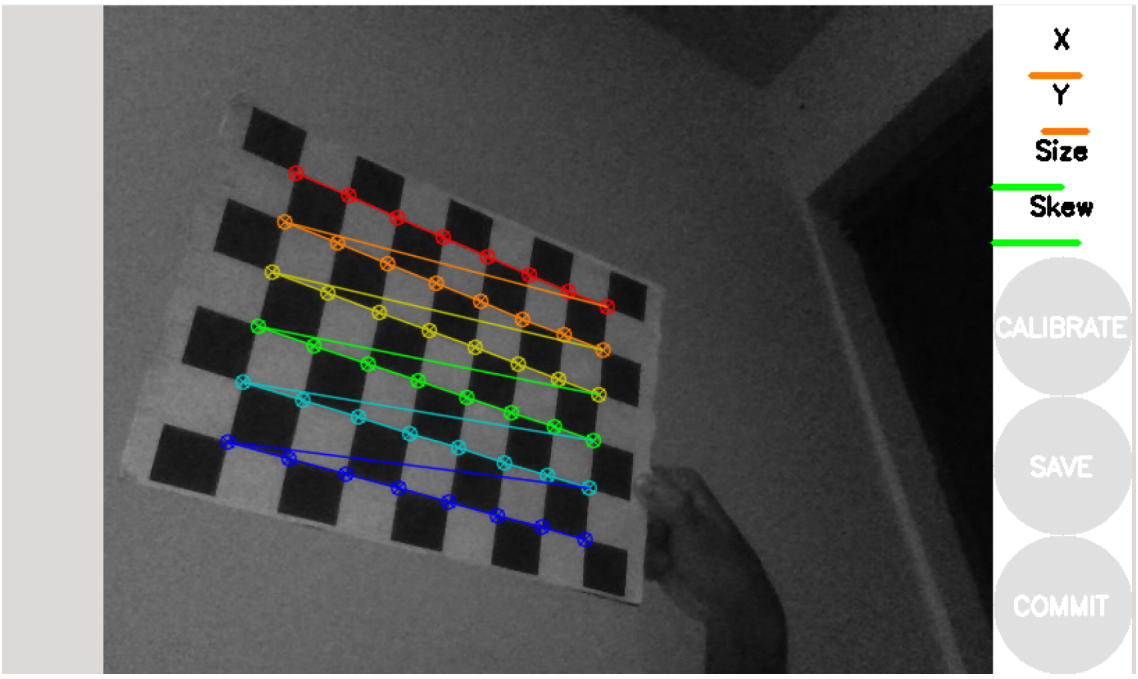

Figura 3.9: Interface do pacote de calibração de câmera.

Com a calibrazação finalizada, a ferramenta retorna duas matrizes, $\boldsymbol{M}$ 3.1 e $\boldsymbol{M_d}$ 3.2, contendo os parâmetros intrísecos da câmera.

$$\boldsymbol{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$\boldsymbol{M_d} = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix} \tag{3.2}$$

Para converter o modelo obtido da câmera pelo pacote do ROS com o modelo que foi utilizado, basta utilizar a equação 3.3 com $r_d = \sqrt{x^2 + y^2}$, conforme [39].

$$\frac{r'}{r} = 1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6 + k_4 r_d^8 + k_5 r_d^{10} \tag{3.3}$$

Porém, para reduzir o custo computacional dentro do nó de localização preferiu-se utilizar o pacote do ROS *image_proc*, a fim de corrigir as distorções da câmera e entregar as imagens já retificadas.

Esse pacote subscreve no *tópico /front/image_raw*, recebendo as imagens da câmera cruas, com distorções, aplica a correção e emite a imagem retificada em um novo *tópico* chamado */front_image/image_rect*, conforme descrito na imagem 3.10

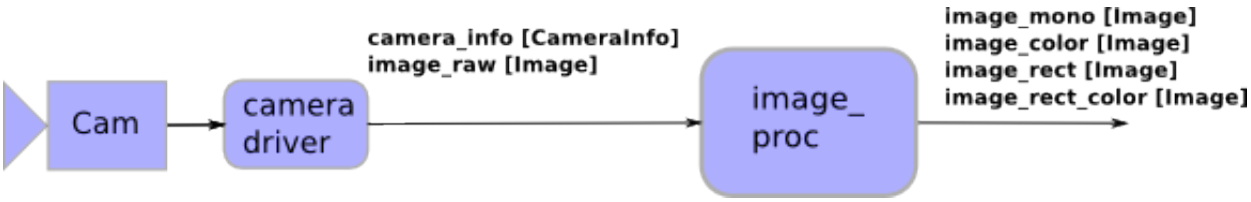

Figura 3.10: Descrição do funcionamento do pacote *image_proc*.

3.3 Localização e Mapeamento Monocular

O processo de localização e mapeamento presente nesse trabalho foi baseado na solução proposta pelo algoritmo PTAM (Parallel Tracking and Mapping), presente em [40] para SLAM baseado em câmeras monoculares. O algoritmo retorna a posição e a orientação da câmera conforme o AR.Drone se movimenta e com esses dados é utilizado um filtro de Kalman Estendido (EKF) para a fusão sensorial com as informações obtidas pelos sensores inerciais do AR.Drone.

3.3.1 *Parallel Tracking and Mapping - PTAM*

O PTAM é um método para a localização e mapeamento visual monocâmera utilizando *keyframes*. Métodos baseados em *keyframes* diferem em técnicas utilizadas em outras abordagens, como, por exemplo, o EKF-SLAM [41]. Porque, ao invés de suprimir as informações da posição da câmera obtidas anteriormente e resumir essas informações em uma matriz de distribuição de probabilidade da posição dos pontos de interesse utilizados como pontos-chave, esse método

utiliza informações de um subconjunto específico de observações anteriores (os *keyframes*) para representar as informações da posição da câmera no sistema de coordenas global com o passar do tempo.

O PTAM possui as seguintes características que o destacam para ser utilizado:

- O processo de mapeamento e o rastreamento de pontos de interesse (*landmarks*) é executado paralelamente em dois processos o que permite a execução em tempo real da aplicação.
- O mapeamento é baseado em *keyframes* e conforme o mapa cresce é aplicado um *Bundle Adjustment* [42], um otimizador numérico baseado em Levenberg-Marquardt para ajustar as posições em que os *keyframes* foram obtidos e os pontos de interesse presentes no mapa conforme o tamanho do mapa aumenta.
- O extrator de pontos de interesse utilizado é o FAST *corners* [10] aplicado em diferentes escalas da imagem.
- Para inicializar o mapa, o PTAM utiliza o algoritmo de 5 pontos [43] com o RANSAC [44] para a obtenção da posição inicial dos pontos do mapa relativa à câmera .

3.3.2 Detecção de Pontos de Interesse

Com o intuito de identificar a relação entre imagens distintas de uma cena é necessário obter alguns pontos de referência em comum entre essas imagens. Esses pontos de referência ou interesse (*keypoint*) são localizados através da identificação de características da cena que não mudem, por exemplo, um gradiente de iluminação, um objeto conhecido ou cantos na imagem.

No PTAM é utilizado o algoritmo FAST (*Features from Accelerated Segment Test*) para a identificação de cantos na cena, sendo que esses cantos são utilizados como pontos de interesse (*landmarks*).

Dado uma imagem se procura por cantos (*corners*), através da busca de características especiais em um dado pixel p presente nessa imagem. O FAST testa se o pixel p contém um canto de acordo com o seguinte procedimento:

- A imagem é passada para a escala de cinza.
- Identifica-se a intensidade luminosa I_p do pixel p em escala de cinza.
- Determina-se um limiar (*threshold*)de diferença na iluminação I_{limiar}.
- Observa-se um círculo de 16 pixels de comprimento ao redor de p.
- p é considerado ser um canto, um ponto de interesse, se existir um conjunto de 12 pixels contínuos ao longo do círculo ao seu redor que possuem itensidades luminosas que sejam maiores do que a itensidade luminosa acima do limiar I_{limiar} $(I_p + I_{limiar})$ ou que sejam menores abaixo do limiar I_{limiar} $(I_p - I_{limiar})$.

A figura 3.11 mostra o funcionamento do procedimento de teste sobre um pixel p. Na imagem vemos um círculo de 16 pixels de perimetro ao redor de p.

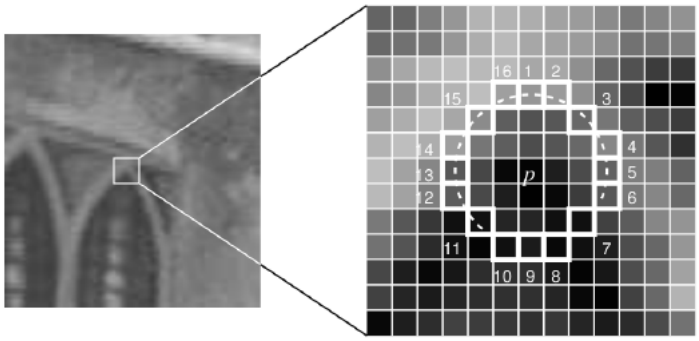

Figura 3.11: Verificação da presença de um canto no pixel p [10].

3.3.2.1 Representação do Mapa

O mapa presente no PTAM possui um conjunto de pontos tridimensionais M, os pontos de interesse (*keypoiknts*), presentes no sistema de coordenadas global O_w. Cada j-ésimo ponto do mapa possui as coordenadas $\boldsymbol{p_{Mj}} = (x_{Mj}\ \ y_{Mj}\ \ z_{Mj})$ e informações da imagem (*keyframe*) em que o ponto foi extraído em escala de cinza. Cada *keyframe* possui a posição e a orientação da câmera frontal do AR.Drone quando a imagem foi obtida e também uma pirâmide de quatro níveis de imagens com versões em escala reduzida da imagem original, conforme a figura 3.12. Por exemplo, o nível-0 possui a imagem original com a sua resolução de 320x240 pixels enquanto o último nível, nível-3, possui a imagem com uma resolução de 40x30 pixels.

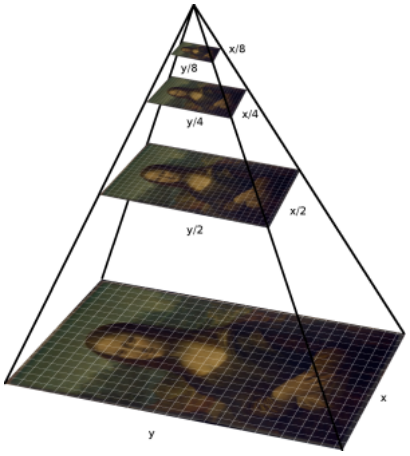

Figura 3.12: Exemplo de uma pirâmide de imagens.

3.3.2.2 O funcionamento do PTAM

O PTAM cria um mapa de pontos tridimensionais utilizando a triangulação de pares de pontos de interesse correspondentes entre diferentes *keyframes*. Ao mesmo tempo o algoritmo utiliza esse

mapa gerado a fim de estimar a posição atual da câmera em relação com o sistema de coordenadas global O_w. O algoritmo possui três diferentes estados, conforme figura 3.13:

- Inicializando.
- Rastreando
- Recuperando

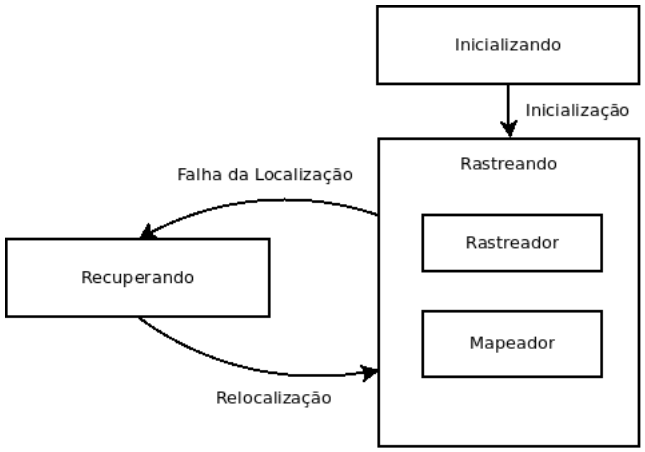

Figura 3.13: Representação do algoritmo PTAM com os seus estados e também os seus processos em paralelo.

No estado de inicialização, um mapa de características da imagem é criado utilizando duas imagens iniciais, obtidas com uma movimentação suave do AR.drone em um plano horizontal.

Ao marcar uma imagem como *keyframe* inicial, o algoritmo FAST *corners* detecta pontos de interesse as características presentes na imagem, pontos de interesse e a regiões ao redor desses pontos. Essas características serão procuradas nos próximos *keyframes* utilizando como métrica o cálculo entre a correlação entre as características.

Ao obter o segundo *keyframe*, todas as características encontradas no primeiro *keyframe* são buscadas no segundo. As característica que possuirem correlação maiores que um determinado limiar são consideradas correspondentes, ou seja, os pontos nas diferentes imagens representam o mesmo ponto no sistema de coordenadas global. Dessa forma, através da homografia entre essas duas imagens, é obtida uma estimativa da posição da câmera utilizando o algoritmo descrito em [45]. Após refinar a estimativa inicial da posição da câmera do AR.drone (em relação com o sistema de coordenadas global) o algoritmo PTAM entra no estado *rastreando*.

No estado *rastreando* é iniciado o processamento paralelo. A primeira *thread*, o rastreador, utiliza informações presentes no mapa para estimar a posição da câmera para o último *keyframe* adicionado ao mapa. A segunda *thread* (o *Mapeador*) mantêm, expande e melhora a qualidade das informações presentes no mapa, conforme novos *keyframes* são adicionados. Uma representação do processo em paralelo dessas duas *threads* está presente na Figura 3.14.

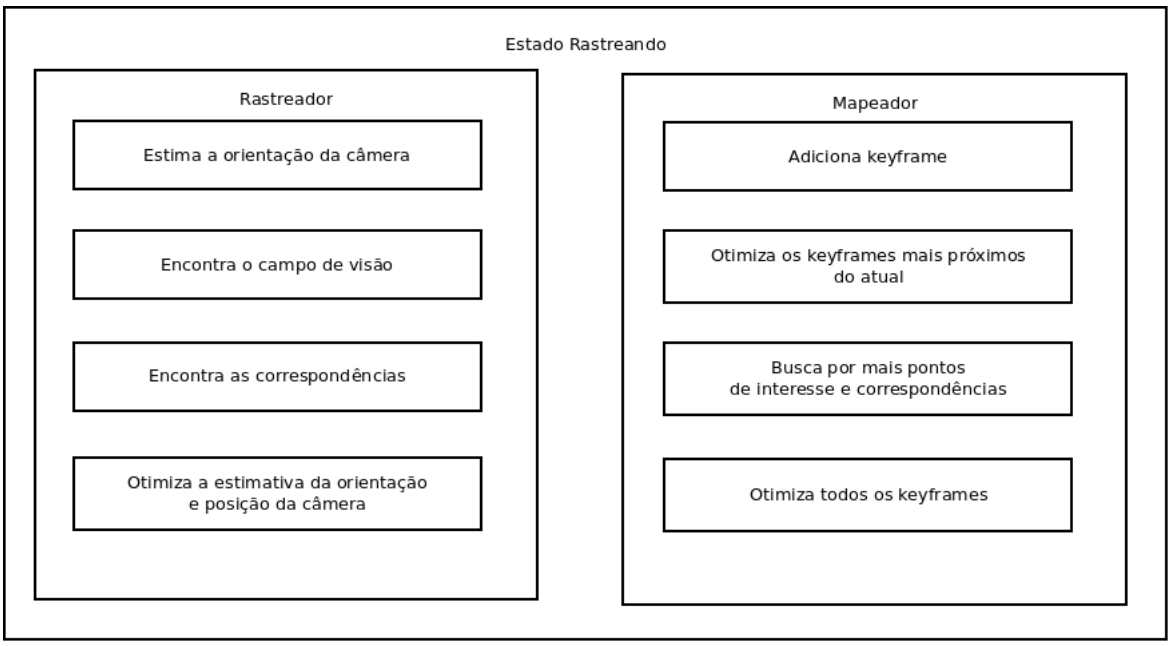

Figura 3.14: Representação das threads presentes no estado rastreando do algoritmo PTAM.

O processo *rastreador* utiliza dois estágios: (a) a estimatição inicial da orientação da câmera, e (b) a estimação refinada da orientação e da posição da câmera do AR.drone.

O primeiro estágio estima a posição da câmera através do algoritmo *Small Blurry Image* [46] que utiliza a pirâmide presente no mapa para cada *keyframe*. A orientação inicial é utilizada como estimativa inicial para o segundo estágio.

O segundo estágio extrai as regiões de interesse e busca no mapa pontos característicos que estão presentes no mesmo campo de visão da orientação obtida do primeiro estágio, visando obter um par de pontos correspondentes e calcular uma estimativa melhor da posição e da orientação da câmera.

O processo mapeador adiciona os novos *keyframes* e utiliza um estimador para encontrar qual é o *keyframe* mais próximo.

O processo mapeador mantêm e expande o conjunto de pontos tridimensionais e o conjunto de *keyframes* presentes no mapa. Quando um novo *keyframe* é adicionado, o processo busca entre os *keyframes* mais próximos para encontrar pares de pontos característicos correspondentes. Esses pares então são triangulados para aumentar o número de pontos do mapa. Logo após isso, é executada uma estimativa entre o *keyframe* atual e os quatro *keyframes* mais próximos, a fim de refinar as posições dos pontos obtidos dessas imagens e, também, refinar a posição da câmera correspondente a essas imagens.

O algoritmo PTAM também possui um processo de recuperação quando uma falha no processo de rastreamento é detectada. O limiar utilizado para definir se o algoritmo deve entrar nesse estado é o número de correspondencias encontradas entre o *keyframe* atual em relação ao *keyframe* mais próximo ser abaixo de um determinado limiar por quinze frames consecutivos.

Essa falha pode ser causada quando o VANT está se movimentando muito rápido, o que causa um efeito de embaçamento na imagem e reduz a eficiência do algoritmo FAST ou se AR.drone mudou demasiadamente o ângulo de guinada. É importante salientar que o PTAM não possui muita robustez quando há grandes variações no ângulo de guinada.

Sem um número suficiente de correspondências entre os *keyframes* presentes no mapa, o processo de rastreamento começa a possuir como resultados posições da câmera incoerentes, e com isso inicia-se um efeito em cadeia em que a posição estimada da câmera se afasta mais e mais do valor real, devido a que a posição do *keyframe* atual depende das posições dos *keyframe* anteriores.

No modo de *recuperação*, o algoritmo para de utilizar informação dos *keyframes* anteriores e começa a utilizar apenas os keyframes obtidos antes do estado de recuperação. O estado de recuperação se mantêm até o limiar de correspondências ficar novamente em um nível aceitável.

3.3.3 Filtro de Kalman Estendido para o SLAM

Os dados de navegação e as leituras da câmera frontal são emitidos pelo AR.Drone em momentos distintos, sendo que as diferenças entre essas leituras também variam, dependendo da qualidade do sinal do WiFi e a porcentagem da bateria, e em casos extremos pode haver a perca dos pacotes com esses dados devido à natureza do protocolo de comunicação UDP.

Para compensar esse problema foi utilizado um EKF. Primeiramente, todos os dados de entrada recebem uma identificação do seu tempo de leitura (amostragem), e os mesmos são salvos em um *buffer*. Então os comandos de controle são calculados utilizando o modelo de predição para o tempo $t + \Delta t_{controle}$.

Essa predição começa no EKF com os dados obtidos no tempo $t - \Delta t_{vis}$, ou seja, desde a última observação da imagem. Depois, é calculada a previsão dos estados do VANT até o tempo $t + \Delta t_{controle}$, utilizando valores das variáveis de controle anteriores e integrando os dados medidos pelos diferentes sensores.

Para o nosso caso, o EKF utiliza o modelo simplificado do quadrirrotor, incluindo o vetor de estados $\boldsymbol{x}_k$, o vetor de entrada de controle $\boldsymbol{u}_k$ e o vetor de medição $\boldsymbol{y}_k$:

$$\boldsymbol{x}_k = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\psi} \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{u}_k = \begin{bmatrix} \bar{\phi} \\ \bar{\theta} \\ \bar{\dot{\psi}} \\ \bar{\dot{z}} \end{bmatrix}^T \tag{3.5}$$

$$\boldsymbol{y}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \\ \phi_k \\ \theta_k \\ \psi_k \\ \dot{\psi}_k \end{bmatrix} \tag{3.6}$$

Para cada diferente fonte de dados é definido uma função de observação (ou função de medição), e a descrição do vetor de medições $\boldsymbol{z}_k$.

A medição de navegação do quadrirrotor corresponde às estimativas das velocidades horizontais do quadrirrotor, os ângulos de arfagem, rolagem e guinada medidos pelos acelerômetros e a estimativa da altitude. Neste caso, as medições do ângulo de guinada são afetadas pelo deslocamento de zero com o tempo e a altitude do VANT é afetada pela presença de variações de altura no chão, e por isso em vez de considerar a medição proveniente dos sensores diretamente para esses dados é considerado a taxa de variação entre eles.

Com isso obtemos a função de medição e o vetor de medições são definidos assim:

$$h_{nav}(\boldsymbol{x}_k) = \begin{bmatrix} \dot{x_k} cos(\psi_k) - \dot{x_k} sen(\psi) \\ \dot{x_k} sen(\psi_k) - \dot{y_k} cos(\psi) \\ \dot{z_k} \\ \phi_k \\ \theta_k \\ \dot{\psi_k} \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{z}_{nav,k} = \begin{bmatrix} \hat{\dot{x}}_k \\ \hat{\dot{y}}_k \\ (\hat{h}_k - \hat{h}_{k-1}) \\ \hat{\phi}_k \\ \hat{\theta}_k \\ (\hat{\psi}_k - \hat{\psi}_{k-1}) \end{bmatrix} \tag{3.8}$$

Caso a medida seja obtida pelo PTAM, considera-se que foi medida a posição e a orientação da câmera do AR.Drone em relação ao sistema de coordenadas global. Neste caso transforma-se

o sistema de coordenadas da câmera para o sistema de coordenadas do quadrirrotor, através de uma transformação constante, tendo em conta que a posição entre a câmera e o centro de massa é conhecida e constante.

Com isso, é obtida a função de medição e o vetor de medições, expressa assim:

$$h_{nav}(\boldsymbol{x}_k) = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \phi_k \\ \theta_k \\ \psi_k \end{bmatrix} \tag{3.9}$$

$$\boldsymbol{z}_{nav,k} = \begin{bmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{z}_k \\ \hat{\phi}_k \\ \hat{\theta}_k \\ \hat{\psi}_k \end{bmatrix} \tag{3.10}$$

3.3.4 Arquitetura do algoritmo de SLAM

A arquitetura do SLAM é baseada nas medições realizadas pelos sensores presentes no AR.Drone e pelas estimações das posições obtidas pelo algoritmo PTAM aplicado à um EKF, para que ocorra a fusão sensorial e as correções necessárias.

Na Figura 3.15 é possível visualizar o algoritmo utilizado para estimar a localização do AR.Drone.

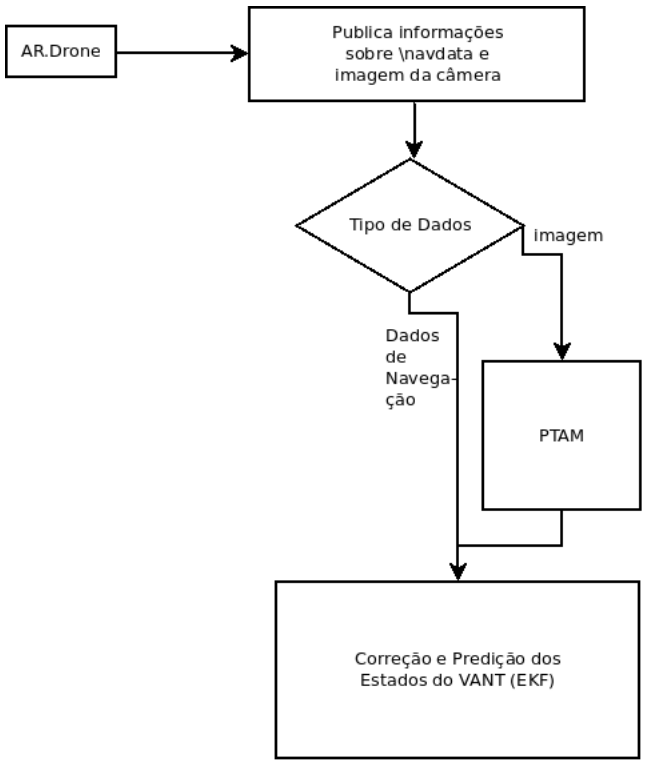

Figura 3.15: Arquitetura do SLAM utilizado.

As aplicações desenvolvidas em C++ no decorrer desse trabalho para a arquitetura apresentada na Figura 3.15 foram: (a) a implementação do filtro de Kalman estendido; (b) o envelopamento dos dados enviados pelo AR.Drone para ficarem compatíveis com as ferramentas presentes na plataforma ROS; (c) o envelopamento dos dados para ficarem compatíveis com as bibliotecas e plataformas utilizadas e os seus algoritmos (PTAM, Qt, OpenCV, Eigen, Gazebo);

3.4 Descrição do Código

3.4.1 Linguagem de Programação

O código foi implementado na linguagem de programação C++. O C++ se destaca como uma linguagem orientada à objetos compilada, com a possibilidade de utilizar recursos de linguagens de programação de nível mais baixo (a linguagem de programação C). Essas características possibilitam o C++ a ter uma boa performance se comparado com outras linguagens de programação de alto nível.

A sua utilização possui como objetivo a implementação de alto nível (orientação à objetos) para sistemas embarcados, com recursos limitados ou desenvolvimento de *software* otimizado. Para a obtenção de um *software* que funcione em tempo real, escolheu-se o desenvolvimento em C++ e também por se tratar de uma linguagem muito utilizada e de fácil portabilidade entre diferentes computadores, aliada a presença de diversas bibliotecas otimizadas para para cálculos númericos, processamento de imagens, etc.

Os compiladores atuais de C++, como, por exemplo, o gcc4.8 (Gnu C Compiler) utilizado, se destaca pela sua eficiência e pela rapidez do binário executável produzido. As penalidades do uso do C++ são, no entanto, o tempo de compilação e também a díficil tarefa de compilar códigos de programas que utilizam recursos de muitas bibliotecas compartilhadas, as quais possuem vários códigos-fontes.

3.4.2 Bibliotecas utilizadas

As bibliotecas utilizadas foram: (a) o OpenCV e o libCVD para implementação de algoritmos de visão computacional e processamento de imagens; (b) o Eigen2 e TooN para a implementação de cálculos que utilizem álgebra linear, matrizes, vetores e estimadores numéricos; e (c) a roslib para a integração entre as aplicações e a ferramenta ROS.

3.5 Utilização da Plataforma SoCKit TERASIC

3.5.1 Instalação do Linux Embarcado na Plataforma SoCKit TERASIC

Para o uso da plataforma foi necessária a instalação de um linux embarcado que dê suporte para o processador baseado na arquitetura ARM presente no SoC, e que também dê suporte para uma instalação funcional do ROS.

Visualizando a página de suporte do ROS é possível verificar que as suas instalações mais estáveis acontecem em sistemas operacionais baseados em Debian/Ubuntu.

Visualizando a página de suporte da Terasic para o kit de desenvolvimentos é possível verificar que há um sistema operacional baseado em uma distribuição Ubuntu Linux desatualizada na página da fabricante e que após algumas manipulações em repositórios a instalação do ROS pode ser efetuada.

Com isso foi necessário gravar a imagem do sistema operacional em um cartão de memória microSD e também configurar os *jumpers* da placa para que o Linux seja selecionado para *boot*. Devido à necessidade de comunicação entre um computador com a placa inicialmente, foi utilizado uma conexão UART através de um cabo USB; mas após a instalação de algumas bibliotecas de rede no linux embarcado a conexão entre o PC e a placa SoC foi uma conexão SSH via cabo ethernet.

Após algumas horas de instalação e alguns *resets* da placa foi possível fazer a alteração e atualização do sistema operacional embarcado no ARM para um Ubuntu 14.04 para a arquitetura Arm7.

3.5.2 Instalação do ROS na plataforma SoC

Para o uso da plataforma foi necessária a instalação do ROS no Linux embarcado. Como o sistema operacional instalado foi o Ubuntu 14.04, que possui uma instalação estável da ROS, a instalação foi feita de forma fácil da ROS.

Primeiramente, foi necessária a instalação das dependências necessárias pela ROS. Depois da instalação dessas dependências o ROS foi instalado com sucesso.

Os códigos para a instalação do sistema operacional Ubuntu na plataforma de desenvolvimento está presente em [47]

3.5.3 Compilação dos códigos da aplicação para o ARM da plataforma SoC

A compilação de algumas bibliotecas necessárias foram bem sucedidas, porém houve alguns problemas de compatibilidade com bibliotecas que já possuíam os dados compilados e por isso não poderiam ser compiladas para a arquitetura da plataforma, outro fator que impossibilitou a continuidade do uso da plataforma SoC como computador remoto, no contexto desse trabalho, foi a instalação do adaptador WiFi via USB que iria ser utilizado pela plataforma para a comunicação com o AR.Drone. A entrada UART to USB do SoC não alimenta o adaptador e por isso o mesmo não poderia ser utilizado e as sucessivas tentativas de ao menos executar a aplicação por hora se mostraram infrutíferas e por isso decidiu-se que seria mais importante a finalização do desenvolvimento da aplicação de localização e mapeamento e que para um projeto futuro seria finalizada a implementação na plataforma SoC que possui muito potencial.

Capítulo 4

Resultados

Esse capítulo tem como o intuito mostrar alguns resultados obtidos com o trabalho a partir de alguns testes.

4.1 Mapeamento e Localização Indoor

Como o AR.Drone não está conseguindo se manter em vôo os testes foram efetuados movimentando o AR.Drone através do laboratório LEIA deixando que a câmera inferior conseguisse visualizar o solo e com isso estimar as velocidades horizontais, assim como para simular uma situação real de voo do AR.Drone. Neste caso, as movimentações foram feitas de forma abrupta.

Como descrito em capítulos anteriores, utilizou-se também do ambiente de simulação do Gazebo com *plugins* para que haja uma simulação do comportamento da aplicação para um AR.Drone em voo.

4.1.1 Visualização Inicial dos Resultados de Localização

Durante esse teste o AR.Drone foi movimentado no ambiente de simulação em frente ao um modelo de uma casa. As movimentações foram suaves e em uma linha reta e ao fim uma movimentação frontal. As imagens nesse teste possuiam muitos *keypoints* em comum.

A Figura 4.1 mostram o AR.Drone voando dentro do ambiente de simulação que foi utilizado.

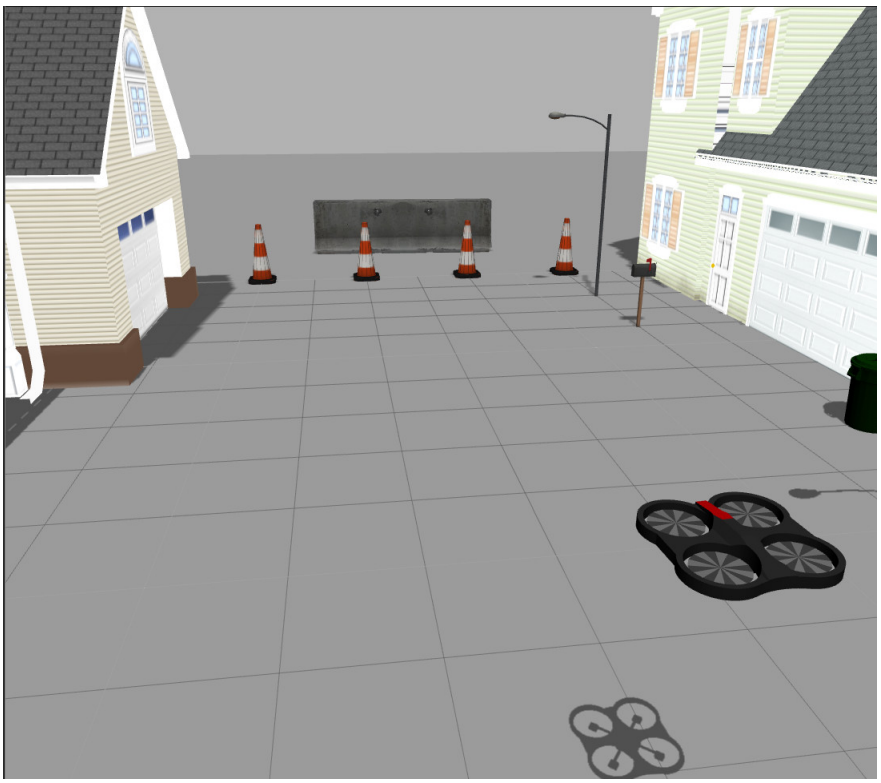

Figura 4.1: AR.Drone simulado dentro de ambiente de simulação Gazebo.

A Figura 4.2 possui uma imagem com a nuvem de pontos obtidas pelo algoritmo PTAM (cruzes em vermelho), a estimação da posição atual do AR.Drone e a sua orientação (eixo de coordenadas maior). Também estão descritas as posições do VANT quando os *keyframes* foram adicionados no mapa (pequenos eixos de coordenadas) e em verde está a trajetória estimada do AR.Drone na cena, utilizando a fusão sensorial dos seus sensores inerciais com a localização visual obtida pelo algoritmo PTAM.

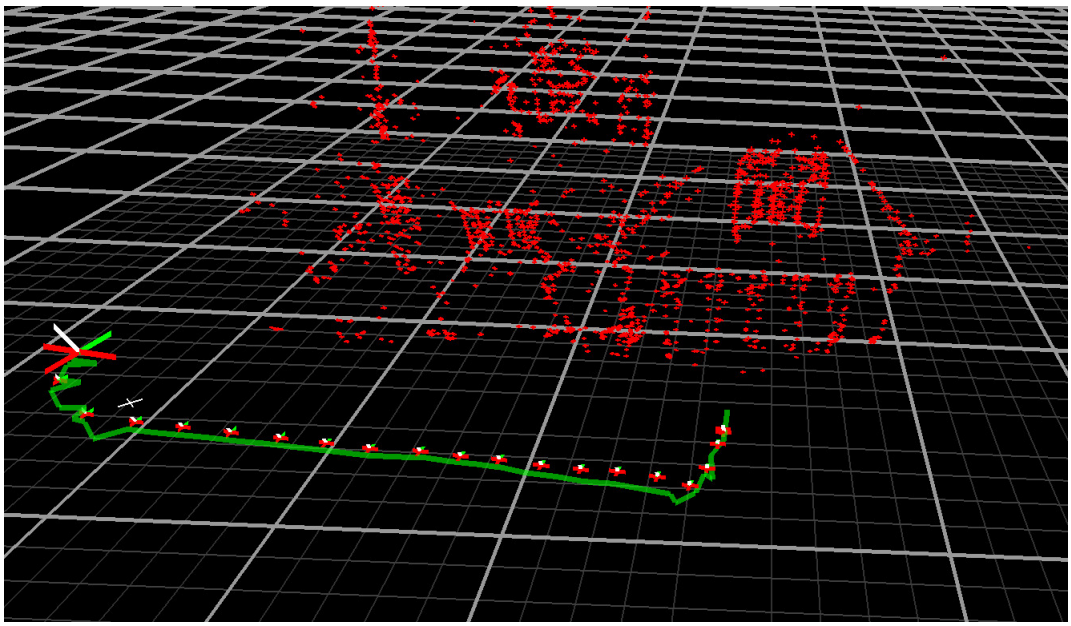

Figura 4.2: Resultados da estimativa do trajeto percorrido pelo AR.Drone no espaço tridimensional utilizando o PTAM com fusão sensorial no ambiente de simulação Gazebo.

A Figura 4.3 possui a última imagem obtida pela câmera, sendo adicionada ao mapa (*key-frame*) uma projeção dos pontos característicos identificados nessa imagem, que também foram encontrados em cenas anteriores presentes no mapa. As cores dos pontos indicam em qual o nível da pirâmide de imagem os *keypoints* foram identificados. Pontos extraídos de níveis mais altos da pirâmide, ou seja, com imagens com baixa escala são representados por pontos azuis, seguidos por pontos verdes, amarelos e, por último, pontos em vermelho, que representam pontos encontrados no nível da pirâmide da imagem original.

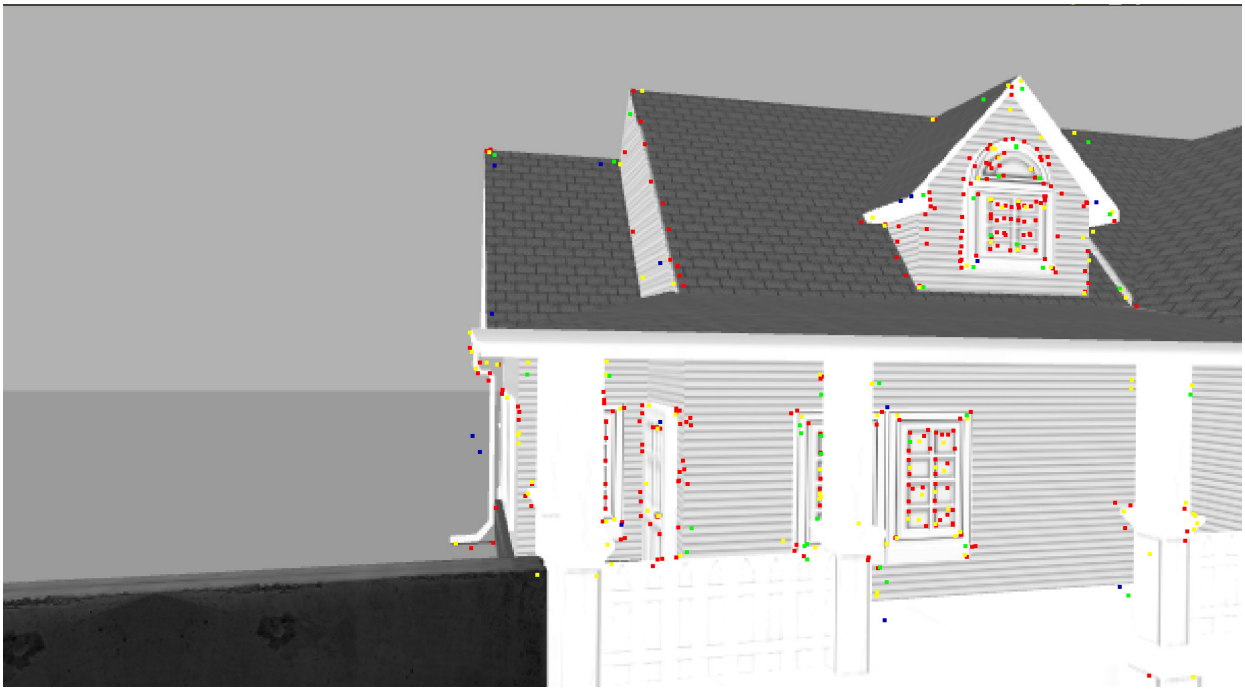

Figura 4.3: Imagem do último keyframe e a projeção dos pontos característicos do mapa que possuem correspondencias com os pontos característicos encontrados na cena atual.

Esse teste mostra o funcionamento e a capacidade de utilização do método aqui descrito. Ao final desse experimento foi obtido um mapa que contém 1757 pontos de interesse distribuídos entre 21 *keyframes*. Através da nuvem de pontos é possível identificar o perfil da casa que estava presente nas imagens, tanto quanto a predição correta do percurso do AR.Drone dentro do ambiente de simulação.

O segundo teste efetuado foi uma prova de estresse no método. Nesse teste foi utilizado o AR.Drone real e o mesmo foi movimentado através do laboratório por um percurso retângular, utilizando movimentos abruptos em suas orientações, com o objetivo de testar a capacidade do algoritmo PTAM de recuperar o rastreamento quando o AR.Drone se movimentava muito rápido, em suas velocidades lineares. Nesse caso há uma forte presença de *motion blur* que afeta o algoritmo, ou quando as imagens recebidas possuem muito poucos (ou nenhum) ponto de interesse em comum com os keyframes obtidos anteriormente.

A Figura 4.4 possui uma imagem com os resultados do método, e.g., nuvem dos pontos de interesse, a estimação da posição e orientação atual do VANT, diferentes posições do VANT quando os *keyframes* foram obtidos pelo PTAM e a trajetória percorrida pelo AR.Drone de acordo com as estimações obtidas pelo EKF.

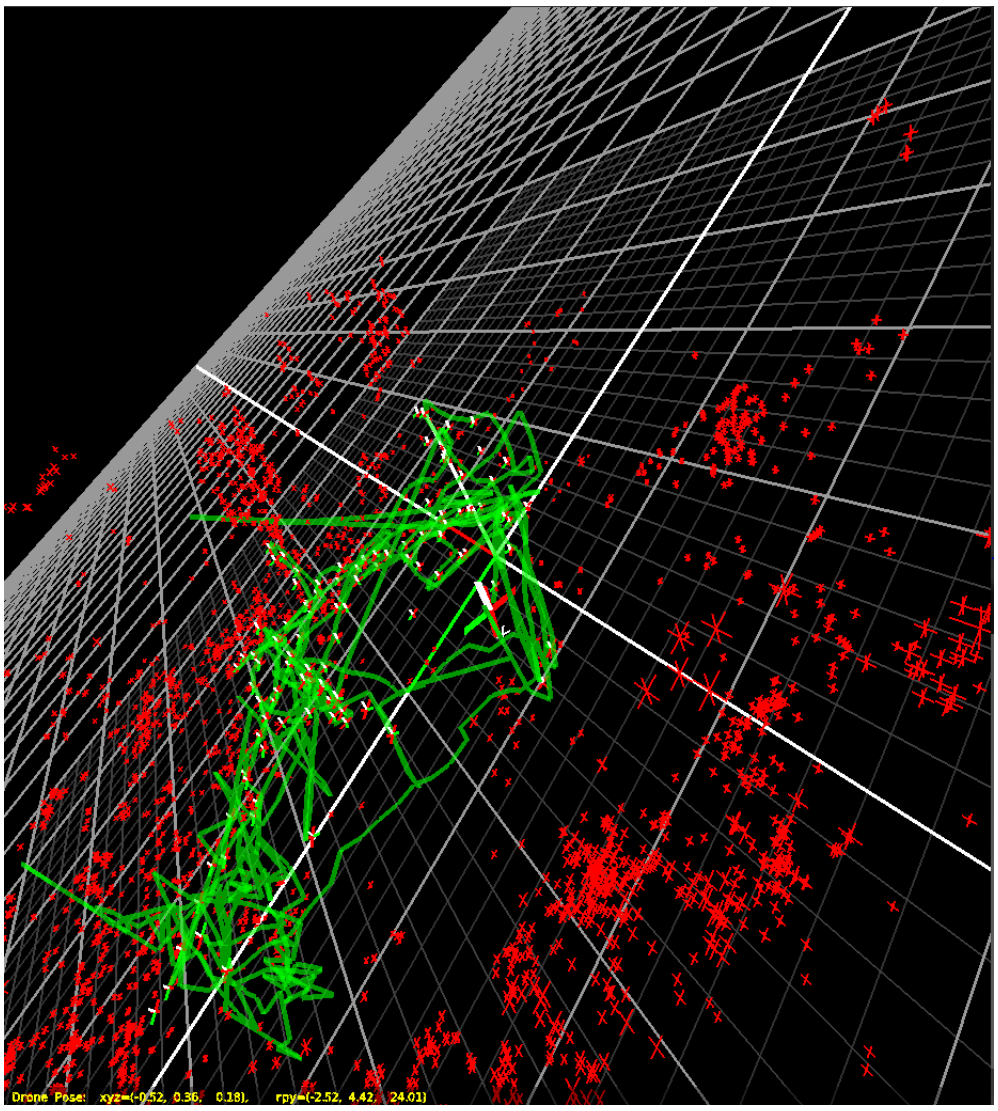

Figura 4.4: Resultados da estimativa do trajeto percorrido pelo AR.Drone no espaço utilizando o PTAM com fusão sensorial.

A Figura 4.5 possui a imagem presente no último *keyframe* do mapa, e também os pontos característicos do mapa que foram encontrados nessa imagem.

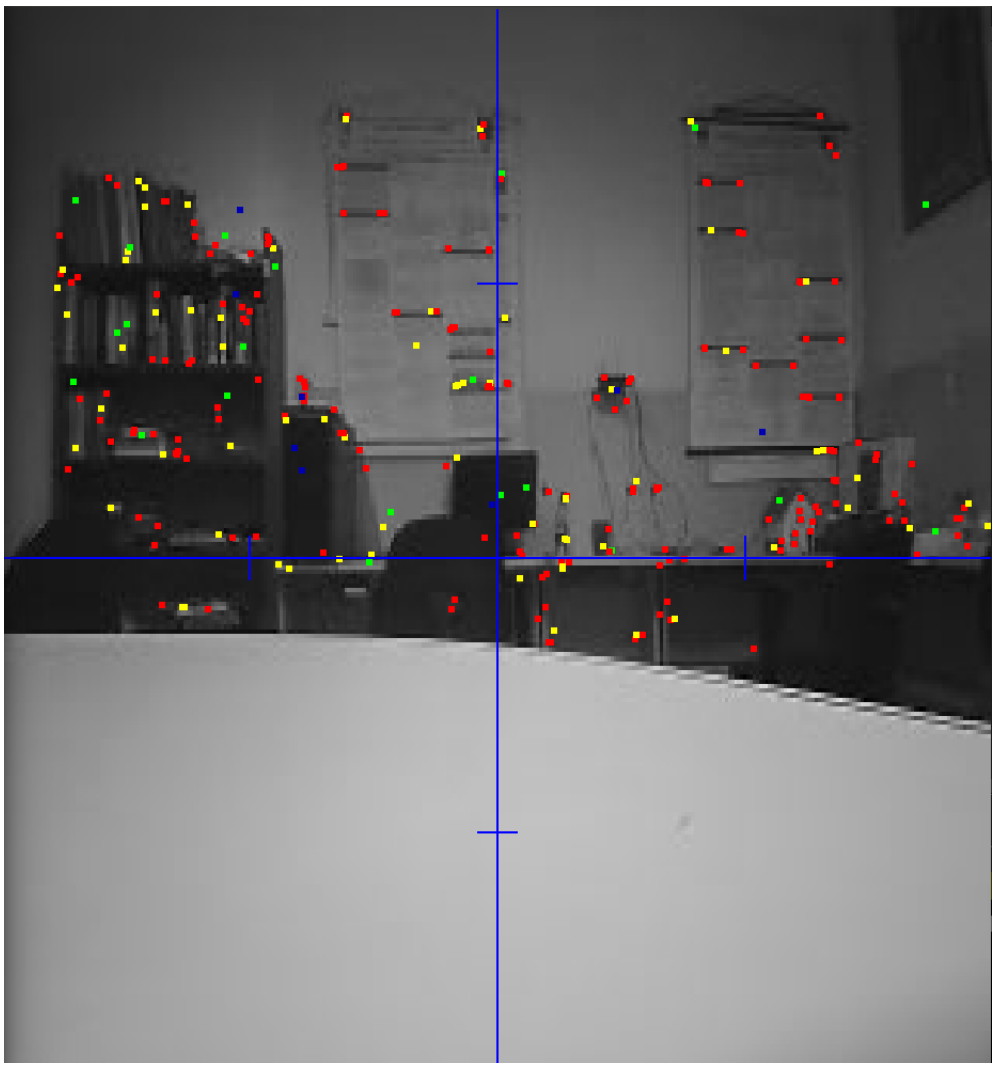

Figura 4.5: Imagem do último *keyframe* e a projeção dos pontos característicos do mapa que possuem correspondências com os pontos característicos encontrados na cena atual.

Considerando que a movimentação do VANT no *plano-xy* foi a de um retângulo com o ponto inicial no vértice inferior a esquerda, é possível visualizar na Figura 4.4 que a movimentação do AR.Drone é semelhante a um formato de um retângulo, porém é possível visualizar que houve vários pontos espúrios na estimação da predição da posição do AR.Drone, sendo por isso que o caminho percorrido mostra-se bastante tortuoso, mas de certa forma esse comportamento era esperado tendo em conta que o caminho foi fechado por duas vezes, sendo que na versão atual a aplicação não trata de fechamento de *loops* no caminho percorrido.

Ao final do teste a aplicação possuia 4165 pontos de interesse no mapa do PTAM e 102 *keyframes*.

4.1.2 Estimativa da dimensão da trajetória a partir da estimação utilizando o filtro de Kalman estedido unido ao PTAM

Nesse teste o AR.Drone (após a etapa de inicialização do PTAM) foi movimentado entre quatro pontos conhecidos do laboratório, a fim de analizar a qualidade da estimação das posições do VANT estimadas.

Nesse experimento o AR.Drone foi colocado em uma cadeira de tal modo que a câmera inferior ainda consiga visualizar o solo para a estimação da velocidade no *plano-xy* pelo *software* embarcado. Porém o sensor de ultrasom ficou obstruído e por isso a estimativa da altura, ou seja, a posição z do AR.Drone, foi ignorada.

O caminho percorrido estimado pelo EKF, após a inicialização do PTAM, está presente na Figura 4.6, sendo que a ordem de movimentalçao foi entre os pontos $A \rightarrow D \rightarrow C \rightarrow B \rightarrow A$.

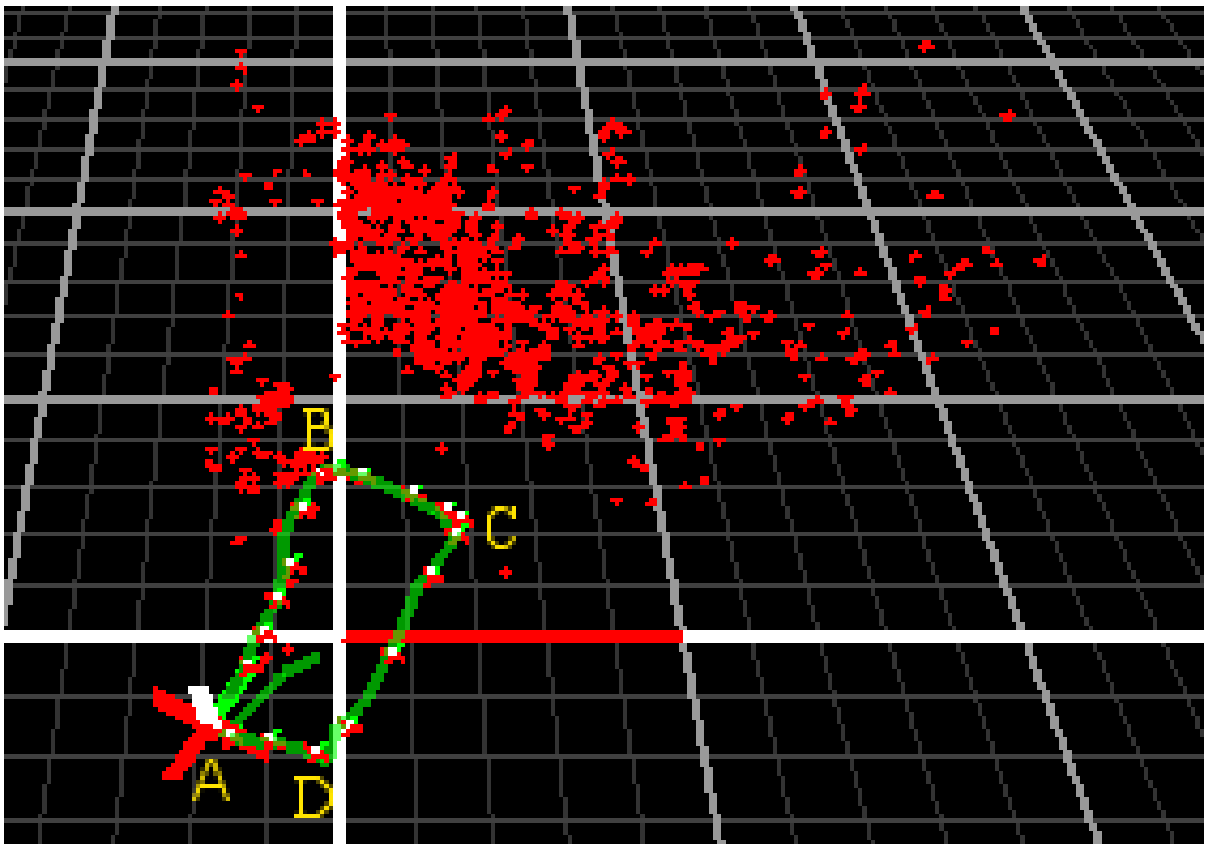

Figura 4.6: Resultado da estimativa do trajeto percorrido pelo AR.Drone.

As dimensões dos lados desse retângulo são $\bar{AB} = \bar{CD} = 1.5$ metros e as dimensões de $\bar{AD} = 0.6$ metros e $\bar{BC} = 0.8$ metros. Essas medidas foram tiradas utilizando-se de uma trena convencional.

As medidas obtidas pela aplicação para esses trechos foram: $\bar{AB} = 1.657$ metros, $\bar{CD} = 1.398$ metros, $\bar{AD} = 0.63$ metros e $\bar{BC} = 0.81$ metros. Os erros para cada uma das medidas comparadas com o valor original foram: $erro(\bar{AD}) = 0.03$ metros (5% do valor original), $erro(\bar{CD}) = 0.102$ metros (6.8% do valor original), $erro(\bar{BC}) = 0.01$ metros (1.25% do valor original) e $erro(\bar{AB}) =$

0.157 metros (10.47% do valor original). .

Neste caso, percebe-se que as estimativas possuem erros que variam bastante entre os comprimentos do retângulo.

Para os lados $\bar{AB}$ e $\bar{CD}$ medidos percebemos que o erro é maior do que as movimentações efetuadas para os lados $\bar{AD}$ e $\bar{BC}$. Isso se deve ao problema da escala da imagem atuando sobre as estimações do EKF, com os dados obtidos pelas estimações do algoritmo PTAM. Conforme o AR.Drone se movimenta na direção da câmera, os objetos de mesmo tamanho são apresentados com tamanhos distintos nas imagens. O tratamento da escala na aplicação é efetuado pelo PTAM e a mesma é estimada durante o processo de homografia da imagem utilizando os pontos de interesse. Para esse experimento as estimativas da escala foram obtidas com sucesso. Muitas vezes apesar do PTAM conseguir preservar o formato da movimentação do AR.Drone a escala para as movimentações em direção ao centro da imagem não representam a escala real do sistema de coordenadas global. Esse problema deverá ser revisitado com maior atenção em trabalhos futuros.

Para verificar a qualidade das etapas de rastreamento do PTAM o mesmo caminho foi executado novamente por dois *loops*, porem utilizando movimentações abruptas no AR.Drone próximos dos pontos conhecidos. Dessa forma obtemos o caminho presente na Figura .

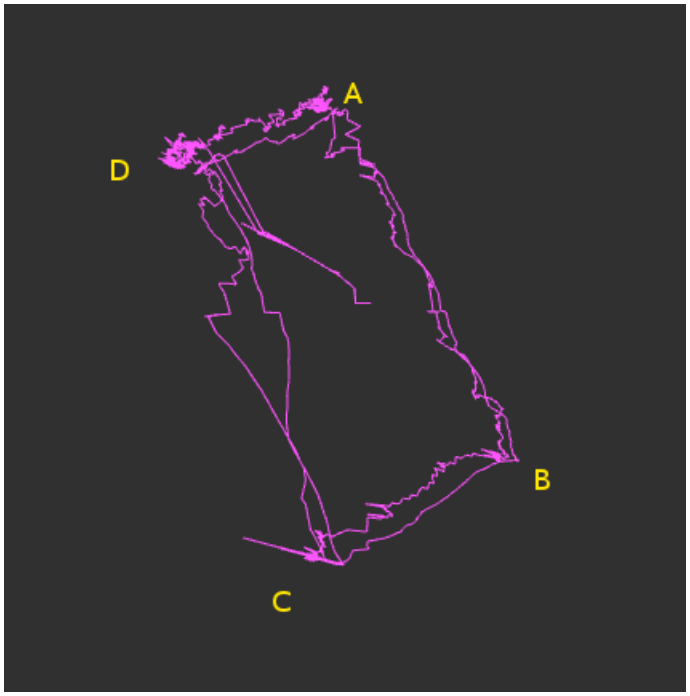

Figura 4.7: Resultado da estimativa do trajeto percorrido pelo AR.Drone entre os pontos A, B, C e D por duas vezes.

Percebe-se, claramente, que durante a execução as estimações da posição nos pontos A e B foram bem ruídosas. Esse efeito é obtido quando o PTAM ainda não possui muitos pontos inseridos no seu mapa e por isso a sua estimação de posição é bastante prejudicada. Também percebe-se que a implementação do EKF é bastante influenciada pelos resultados obtidos pela estimação visual da posição obtido pelo PTAM e conforme as estimativas do PTAM vão se tornando melhores (aumento

no número de pontos característicos no mapa e aumento do número de correspondências entre os *keyframes*).Neste caso, temos que a qualidade da estimação do trajeto percorrido é bem melhor.

4.1.3 Verificação da Utilização dos Recursos de Hardware

Nesse experimento a intenção foi verificar a utilização do *hardware* do computador remoto quando todas partes do programa estão sendo utilizadas ao mesmo tempo: o algoritmo PTAM junto com o EKF, a interface gráfica do *drone_gui* e o simulador do Gazebo.

Na Figura 4.8 é possível verificar entre o período de 60 à 30 segundos antes do tempo atual a utilização do sistema operacional dos recursos de hardware da máquina antes da inicialização do Gazebo. A máquina utilizada foi um PC com 8Gb de memória RAM e processador Intel i7 4710MQ [**?**] com sistema operacional Ubuntu Linux 14.04. Os resultados desse experimento mostram que apesar de nenhum programa ativo no momento existem alguns poucos processos sendo executados em *background* e esses são processos inerentes ao sistema operacional. A utilização de memória no entanto já é grande (cerca de 60% da memória RAM disponível). Isso se deve às inicializações anteriores dos programas, e que é grande a necessidade de uma otimização na desalocação de memória pelas aplicações porque esse uso anormal de memória é causado por uma grande quantidade de *memory leakage* (vazamento de memória).

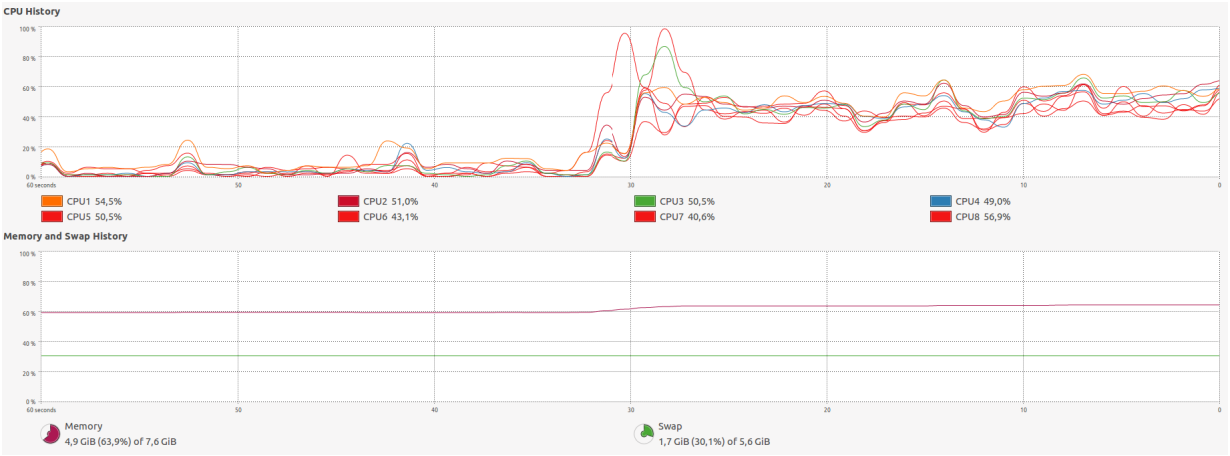

Figura 4.8: Utilização dos recursos de hardware do computador remoto utilizando apenas o ambiente de simulação do Gazebo.

Com a inicialização do Gazebo é fácil visualizar o aumento repentino do uso dos processadores. Isso demonstra que o ambiente de simulação Gazebo já é um programa que utiliza bastante recursos de máquina porque é responsável por simular um ambiente tridimensional com diversos modelos que podem colidir entre si e que possuem uma dinâmica de movimentação própria, como, por exemplo, o AR.Drone. Além disso há a necessidade de renderizar esses modelos tridimensionais, visualizá-los dependendo da posição e orientação atual da câmera e outros recursos de ambientes de simulação. A utilização de recursos de CPU ficam variando entre 40 e 60% e a utilização de memória só aumentou 4%.

Na Figura 4.9 o computador está utilizando todas as tarefas com um mapa obtido pelo PTAM com 4509 pontos e 111 cenas (vide Figura 4.10).

Percebe-se que a utilização do computador é máxima e que apesar do desenvolvimento da aplicação utilizar recursos de processamento paralelo a quantidade de recursos utilizados é muito grande. Isso se deve porque o algoritmo do PTAM possui custo computacional dependente do tamanho do mapa e também porque conforme o tempo de simulação avança o Gazebo começa a consumir mais recursos de *hardware*. Esse problema é agravado ainda mais, conforme os relatos presentes no fórum da aplicação [48], para sistemas que utilizam placas gráficas Intel Graphics e, atualmente, o computador remoto utiliza uma Intel Chipset Integrated Graphics Controller e isso, infelizmente, é um dos *known issues* atuais do Gazebo.

Em algoritmos de SLAM o aumento no custo computacional conforme o mapa aumenta é um dos principais fatores que limitam a utilização em tempo real de técnicas de SLAM.

Considerando que o uso de recursos da máquina já é grande e a presença de muitos *nós* (que são apenas de simulação) considera-se que no caso de cenas pequenas é possível utilizar a aplicação utilizando um computador remoto com menos recursos computacionais. O único fator limitante é a memória RAM utilizada que deve ter o seu uso otimizado.

Figura 4.9: Utilização dos recursos de hardware do computador remoto utilizando o ambiente de simulação do Gazebo aliado aos nós de SLAM do AR.Drone e da interface gráfica. Com o mapa presente no PTAM saturado com 4509 pontos de interesse.

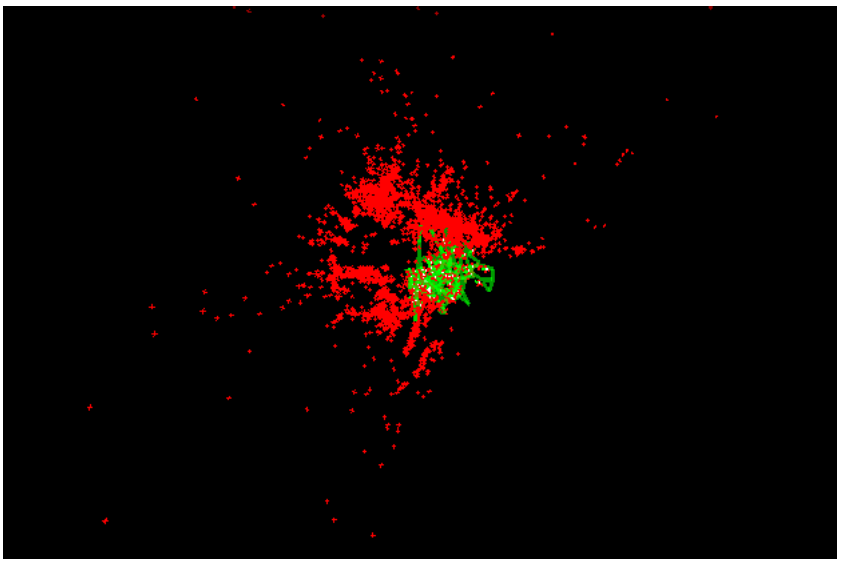

Figura 4.10: Mapa presente no PTAM representando uma cena grande com 4509 pontos de interesse. Percebe-se que devido a escala aumentada da cena e os diversos pontos espúrios durante a execução é difícil visualizar o caminho percorrido pelo AR.Drone.

Na Figura 4.11 o computador remoto está executando apenas as aplicações do PTAM junto com o pacote do ROS *ardrone_ driver* e percebe-se que para um mapa convencional de 1480 *keypoints* e 43 *keyframes* a quantidade de recursos utilizados pela aplicação é bem menor com os diferentes processos utilizando menos tempo de CPU, que variam entre 0 à 20%, exceto em períodos de inicialização do mapa onde é grande o uso de um processo da CPU e que acontece antes do algoritmo PTAM executar concorrentemente.

Já a utilização de memória ainda se mostra bastante defeituosa (quase 80% da memória disponível sendo utilizada) e que futuramte é necessário que haja uma atenção especial na desalocação de memória pela aplicação. Esse aumento de memória comparada ao último teste se deve à utilização de um navegador de internet aberto em *background*.

No futuro espera-se executar essa aplicação utilizando a placa SoCKit como computador remoto e por isso é importante saber a utilização de recursos de hardware pela aplicação.

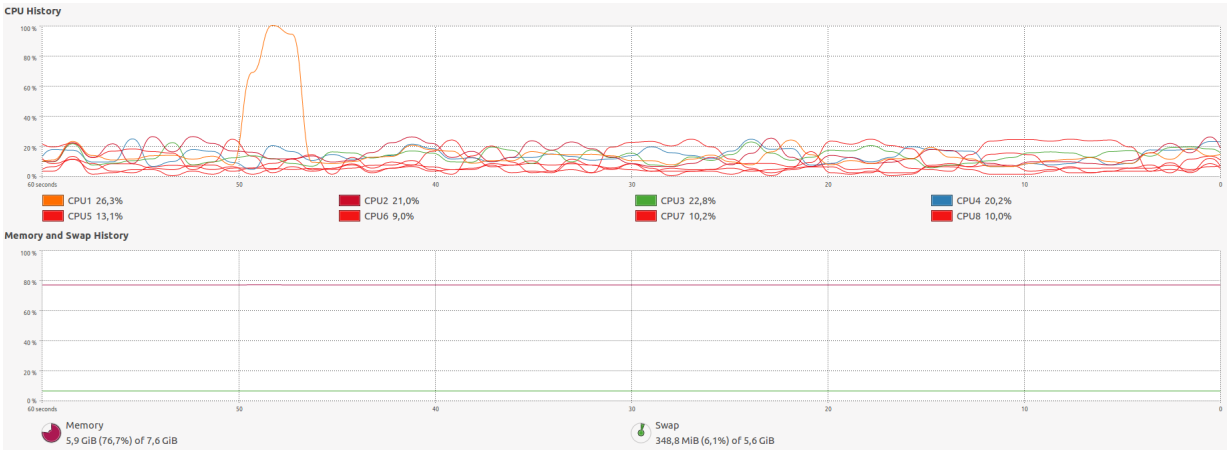

Figura 4.11: Utilização dos recursos de hardware do computador remoto utilizando o nó de SLAM do AR.Drone e o pacote *ardrone_driver*. Com o mapa presente no PTAM com 1480 pontos de interesse.

Na Figura 4.12 está a visualização do mapa obtido obtido nesse experimento.

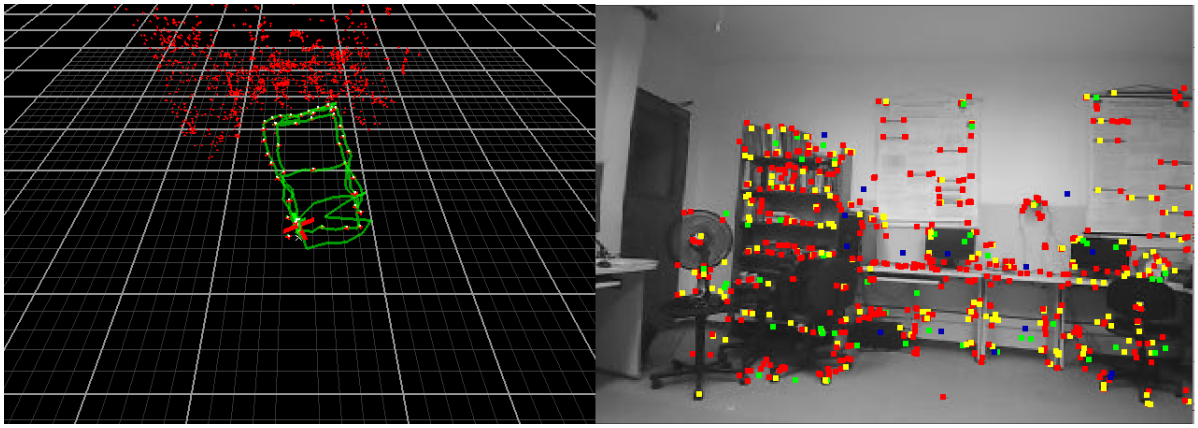

Figura 4.12: Mapa presente no PTAM representando uma cena com 1480 pontos (esquerda). Imagem representando o último *keyframe* do mapa com os pontos de interesse encontrados na imagem (direita).

4.2 Sumário dos Resultados

Ao final do trabalho desenvolvido foi possível obter a localização e o mapeamento simultâneo de um VANT em dois cenários distintos: (a) utilizando o VANT AR.Drone se comunicando via WiFi com um computador remoto (PC) dentro de um ambiente fechado (laboratório) e (b) utilizando um ambiente de simulação com um modelo de um quadrirrotor executado em um PC.

Apesar de não utilizar a placa SoCKit como computador remoto durante os testes dos métodos de localização e mapeamento desenvolvidos para o VANT. Uma das contribuições desse trabalho foi a instalação bem sucedida da ROS na placa de desenvolvimento SoCKit, que, com certeza, irá ser utilizada em trabalhos futuros.

A Figura 4.13 representa de forma simples a arquitetura do sistema implementado para testes utilizando o AR.Drone e a Figura 4.14 possui uma representação simples a arquitetura do sistema implementado para testes utilizando o ambiente de simulação.

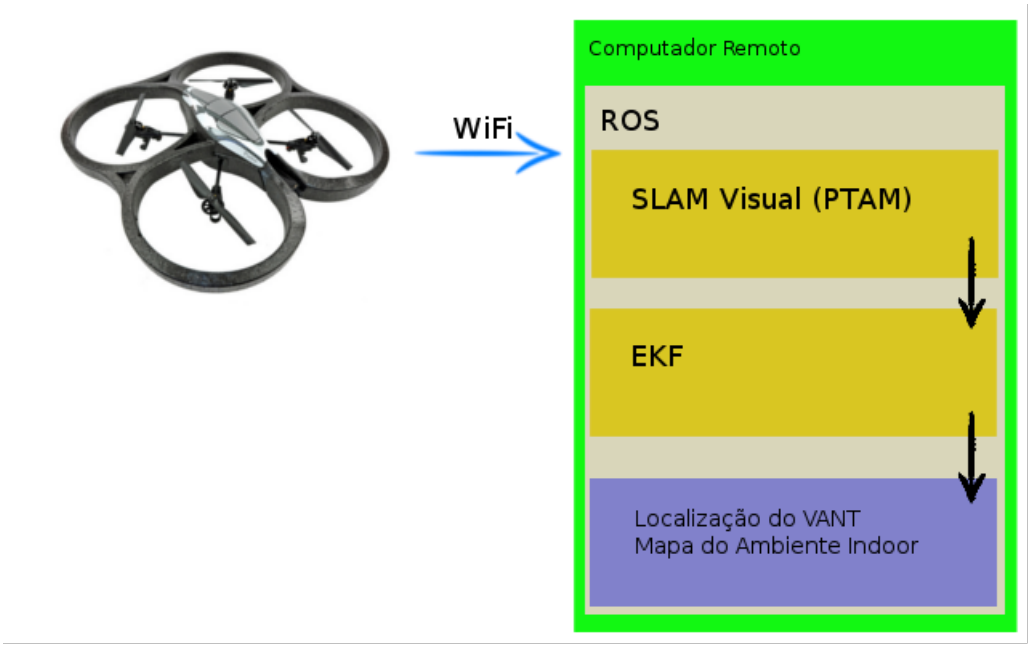

Figura 4.13: Arquitetura do sistema implementado utilizando um AR.Drone se comunicando via WiFi com um computador remoto, que utilizando os dados dos sensores e câmera frontal do AR.Drone obtem um mapa do ambiente e a localização atual do AR.Drone neste ambiente.

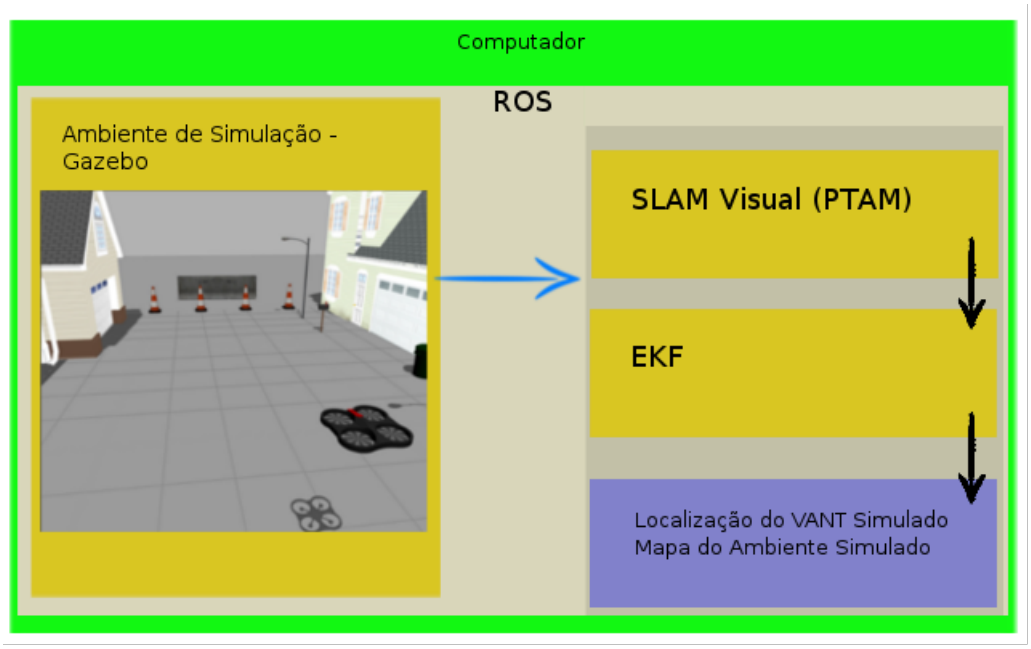

Figura 4.14: Arquitetura do sistema implementado utilizando o ambiente de simulação Gazebo no computador remoto para obter um mapa do ambiente simulado e a localização atual do quadrirrotor dentro da simulação.

Capítulo 5

Conclusões

5.1 Comentários Finais

Como todo trabalho de robótica a utilização do *harware* pode tornar-se rapidamente em um problema de difícil solução. Isto é devido a que todo o desenvolvimento do trabalho é bastante dependente do comportamento e do bom estado do hardware do robô. No caso desse trabalho, infelizmente, apesar das diversas tentativas infrutíferas e a constante pesquisa em forúns do fabricante e descrições de outros usuários, o quadrirrotor utilizado mostrou-se, para todos os fins práticos, inoperante para voo.

No entanto, para contornar esse problema sério utilizou-se de intensa pesquisa de soluções para encontrar ferramentas e meios em que o trabalho apesar de prejudicado pudesse ser finalizado. O resultado dessa pesquisa deu-se em forma na utilização do Gazebo para simular o comportamento do VANT em voo.

Por ser um trabalho em mecatrônica, buscou-se utilizar de recursos e conceitos aprendidos durante todo o decorrer do curso nesse trabalho.

A pesquisa sobre o *hardware* do VANT e o seu funcionamento mostram o estado da arte atualmente, para o desenvolvimento de robôs aéreos e que nos próximos anos espera-se que as aplicações do uso desse tipo de plataforma seja muito maior.

O modelamento do quadrirrotor possui natureza bastante não linear e durante a sua simplificação utilizou-se recursos simples que se mostraram bastante eficazes.

A implementação de técnicas de localização e mapeamento mostraram-se bastante desafiadoras e utilizam-se de princípios que são fundamentais na visão computacional e no uso de sistemas de medições.

Como os algoritmos de extração de pontos de interesse possuem bastante custo computacional foram testadas algumas heurísticas para encontrar os parâmetros dos algoritmos utilizados que atendessem melhor a implementação do sistema de localização.

Outro problema que não é informado pelo fabricante é o fato que o *hardware* embarcado no

AR.Drone 1.0 não disponibiliza (através do seu canal de comunicação) a imagem original, mas uma versão com escala reduzida pela metade, e depois de ter sido utilizado uma compressão de imagem com perda, sendo que com isso a imagem obtida possui alguns artefatos causados pela compressão. Esse fato dificulta o uso eficiênce de alguns algoritmos de detecção de parâmetros da imagem, aumentando o número de pontos de interesse falsos encontrados na imagem.

As aplicações desenvolvidas utilizando recursos das ferramentas do ROS foram executados em uma máquina com processador i7 com 4 núcleos e 8 *threads* de 2.5 GHz com 8 GB de memória RAM. Isso tornou possível a utilização de recursos de simulação do Gazebo aliado ao uso da aplicação desenvolvida.

O uso de mapas por *octomap* [49], começou a ser implementado, mas foi interrompido porque a quantidade pontos obtidos durante o mapeamento não é grande o suficiente para justificar o uso, e também da forma como a implementação foi feita na aplicação. Apesar da eficiência excepcional do *octomap*, o processo de rastreamento na aplicação do PTAM tornou-se inesperada e instável, conforme a núvem de pontos era publicada em um tópico para cada *keyframe* adicionado ao mapa. Porém, para projetos futuros com esse tema espera-se a integração da aplicação com a plataforma *octomap*.

A utilização da plataforma SoCKit mostrou-se no início do desenvolvimento do trabalho como uma abordagem eficaz para a implementação dos métodos com bastante custo computacional em tempo real. Porém, devido aos problemas mencionados no capítulo 3 preferiu-se que para que o trabalho seja implementado a tempo para a apresentação que a utilização do SoCKit fique para trabalhos futuros.

5.2 Perspectivas Futuras

Para a continuação com o trabalho dessa linha de pesquisa sugere-se:

- Encontrar um meio de reparar o AR.Drone ou substituí-lo por um novo VANT, como por exemplo, o AR.Drone 2.0 que possui melhorias nos sensores e também no hardware e software embarcados.
- Desenvolvimento de um controlador que utiliza as técnicas de modelo de controle preditivo para o sistema de espaço de estados utilizado do quadrirrotor.
- Realizar mais experimentos para validar a utilização da localização e mapeamento apresentado.
- Desenvolver um estimador melhor para a escala do percurso percorrido conforme o VANT se locomove.
- Utilizar um adaptador WiFi USB que seja alimentado externamente na plataforma SoC.
- Encontrar os binários de algumas bibliotecas compartilhadas para o ARM, ou compilar elas com um compilador cruzado para o ARM da SoCKit.

- Utilizar a mesma aplicação utilizando o SoCKit como computador remoto.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] PREDATOR UAV, United States of America. http://www.airforce-technology.com/projects/predator-uav/. Acessado em 09/07/2016.

[2] JANNA, W. *Introduction to fluid mechanics*. PWS Engineering, 1987. ISBN 9780534076320. Disponível em: <https://books.google.com.br/books?id=P6o-AQAAIAAJ>.

[3] MD4-1000. https://www.microdrones.com/en/products/md4-1000/. Acessado em 09/07/2016.

[4] AR.DRONE 2.0 Technical Specifications. https://www.parrot.com/fr/drones/parrot-ardrone-20-elite-%C3%A9dition#technicals. Acessado em 25/09/2016.

[5] NASA Updates on Mars Exploration Missions. https://www.newscientist.com/article/dn17351-nasa-readies-sandbox-to-plot-mars-rovers-escape/. Acessado em 25/09/2016.

[6] AR.DRONE Motor Replacement. https://www.ifixit.com/Guide/Parrot+AR.Drone+Motor+Replacement/36033. Acessado em 09/07/2016.

[7] BRISTEAU, P.-J. et al. The navigation and control technology inside the ar. drone micro uav. *IFAC Proceedings Volumes*, Elsevier, v. 44, n. 1, p. 1477–1484, 2011.

[8] MATHWORKS - Documentation: What Is Camera Calibration? https://www.mathworks.com/help/vision/ug/camera-calibration.html. Acessado em 20/11/2016.

[9] WELCH, G.; BISHOP, G. An introduction to the kalman filter. 1995.

[10] ROSTEN, E.; DRUMMOND, T. Machine learning for high-speed corner detection. In: *European Conference on Computer Vision*. [s.n.], 2006. v. 1, p. 430–443. Disponível em: <http://www.edwardrosten.com/work/rosten$_2$006$_m$*achine.pdf*>.

[11] ASIMOV, I. *I, Robot*. Bantam Books, 1950. (Robot series). Disponível em: <https://books.google.com.br/books?id=MD0GAQAAIAAJ>.

[12] VACUUMING Robot - iRobot Roomba. http://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx. Acessado em 08/07/2016.

[13] GOOGLE Self-Driving Project. https://www.google.com/selfdrivingcar/. Acessado em 08/07/2016.

[14] TESLA Model S Autopilot- Model S Software Version 7.0 Features. https://www.teslamotors.com/en_AU/presskit/autopilot. Acessado em 08/07/2016.

[15] UAVS FOR CIVIL SECURITY. https://www.microdrones.com/en/applications/areas-of-application/security/. Acessado em 08/07/2016.

[16] DRONE-BASED INDUSTRIAL INSPECTIONS. https://www.microdrones.com/en/applications/areas-of-application/inspection/. Acessado em 08/07/2016.

[17] UNMANNED CARGO SYSTEM. https://www.microdrones.com/en/applications/areas-of-application/unmanned-cargo-system/. Acessado em 08/07/2016.

[18] NOTíCIA: Monitoramento de focos de dengue utilizando drones. http://g1.globo.com/sao-paulo/noticia/2016/02/sp-usara-drone-para-monitorar-foco-de-dengue-e-app-para-envio-de-
html. Acessado em 15/12/2016.

[19] PARROT AR.Drone Review on a blog. http://www.generationrobots.com/blog/en/2016/03/ros-robot-operating-system-2/. Acessado em 25/09/2016.

[20] KRAJNÍK, T. et al. Ar-drone as a platform for robotic research and education. In: SPRINGER BERLIN HEIDELBERG. *International Conference on Research and Education in Robotics*. [S.l.], 2011. p. 172–186.

[21] EPP, Expanded Polypropylene. http://www.epp.com/. Acessado em 09/07/2016.

[22] PERKINS, C. E. et al. *Ad hoc networking*. [S.l.]: Addison-wesley Reading, 2001.

[23] BMA150 Digital, triaxial acceleration sensor Datasheet. http://odroid.com/dokuwiki/lib/exe/fetch.php?media=en:universal_motion_joypad:bma150.pdf. Acessado em 09/07/2016.

[24] BORENSTEIN, J.; KOREN, Y. Obstacle avoidance with ultrasonic sensors. *IEEE Journal on Robotics and Automation*, IEEE, v. 4, n. 2, p. 213–218, 1988.

[25] LUCAS, B. D.; KANADE, T. et al. An iterative image registration technique with an application to stereo vision. In: *IJCAI*. [S.l.: s.n.], 1981. v. 81, n. 1, p. 674–679.

[26] SCHUNCK, B. "determining optical flow": a retrospective. *Artificial Intelligence*, v. 59, p. 81–87, 1993.

[27] ROS, Robotic Operating System. http://www.ros.org/. Acessado em 09/07/2016.

[28] SOCKIT - the Development Kit for New SoC Device. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=816. Acessado em 09/07/2016.

[29] BRESCIANI, T. Modelling, identification and control of a quadrotor helicopter. *MSc Theses*, 2008.

[30] RICH, M. *Model development, system identification, and control of a quadrotor helicopter*. Tese (Doutorado) — Iowa State University, 2012.

[31] ENGEL, J.; STURM, J.; CREMERS, D. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. In: *In Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*. [S.l.: s.n.], 2012. v. 320.

[32] HARTLEY, R.; ZISSERMAN, A. *Multiple view geometry in computer vision*. [S.l.]: Cambridge university press, 2003. 153-238 p.

[33] KALMAN, R. E. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, American Society of Mechanical Engineers, v. 82, n. 1, p. 35–45, 1960.

[34] HOW to replace the AR.drone propellers. `http://the-parrot-ardrone.com/ar-drone-spares-repairs/how-to-replace-the-ar-drone-propellers/`. Acessado em 25/11/2016.

[35] GAZEBO - Robot simulation made easy. `http://gazebosim.org/`. Acessado em 10/11/2016.

[36] TUM AR.Drone Simulator. `http://wiki.ros.org/tum_simulator`. Acessado em 10/11/2016.

[37] MEYER, J. et al. Comprehensive simulation of quadrotor uavs using ros and gazebo. In: SPRINGER. *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. [S.l.], 2012. p. 400–411.

[38] ZHANG, Z. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 11, p. 1330–1334, 2000.

[39] DEVERNAY, F.; FAUGERAS, O. D. Automatic calibration and removal of distortion from scenes of structured environments. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *SPIE's 1995 International Symposium on Optical Science, Engineering, and Instrumentation*. [S.l.], 1995. p. 62–72.

[40] KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small ar workspaces. In: IEEE. *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. [S.l.], 2007. p. 225–234.

[41] JUAN, J. A. C. J. N.; TARDÓS, D. Limits to the consistency of ekf-based slam. Citeseer.

[42] TRIGGS, B. et al. Bundle adjustment—a modern synthesis. In: SPRINGER. *International workshop on vision algorithms*. [S.l.], 1999. p. 298–372.

[43] STEWÉNIUS, H.; ENGELS, C.; NISTÉR, D. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 60, p. 284–294, jun. 2006. Disponível em: <http://dx.doi.org/10.1016/j.isprsjprs.2006.03.005>.

[44] ZULIANI, M. Ransac for dummies. *Vision Research Lab, University of California, Santa Barbara*, Citeseer, 2009.

[45] FAUGERAS, O. D.; LUSTMAN, F. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence*, World Scientific, v. 2, n. 03, p. 485–508, 1988.

[46] BENHIMANE, S.; MALIS, E. Homography-based 2d visual tracking and servoing. *The International Journal of Robotics Research*, SAGE Publications, v. 26, n. 7, p. 661–676, 2007.

[47] CODIGO de Instalação: ARM SOC Kit. `https://github.com/jessebarreto/ROSberryPi_Dev`. Acessado em 15/12/2016.

[48] GITBUCKET OSRF: Gazebo - Known Issues. `https://bitbucket.org/osrf/gazebo/issues?q=intel+graphics`. Acessado em 01/12/2016.

[49] HORNUNG, A. et al. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, Springer, v. 34, n. 3, p. 189–206, 2013.

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

No CD entregue com o trabalho está sendo disponibilizada uma cópia digital do deste relatório e também uma cópia com todos os arquivos utilizados para elaboração desse trabalho. Além disso, o arquivo também contem os códigos-fonte utilizados um README com as instruções de compilação e execução dos códigos desse trabalho para serem executados.